Software Productivity Consortium

Domain Analysis and Design

SPC-96019-CMC
Version 01.00.05

May 1996

SOFTWARE
PRODUCTIVITY
CONSORTIUM

SPC Building
2214 Rock Hill Road
Herndon, Virginia 22070
(703) 742-8877

Domain Analysis and Design
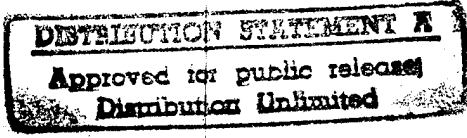
SPC-96019-CMC

Version 01.00.05

May 1996

19960612 003

Produced by the
SOFTWARE PRODUCTIVITY CONSORTIUM

SPC Building
2214 Rock Hill Road
Herndon, Virginia 22070

DTIC QUALITY INSPECTED 3

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE May 1996	3. REPORT TYPE AND DATES COVERED Technical Report - Final

4. TITLE AND SUBTITLE Domain Analysis and Design	5. FUNDING NUMBERS
6. AUTHOR(S) Produced by Software Productivity Consortium under contract to DARPA	G MDA972-92-J-1018

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES) Software Productivity Consortium SPC Building 2214 Rock Hill Road Herndon, VA 22070	8. PERFORMING ORGANIZATION REPORT NUMBER SPC-96019-CMC, Version 01.00.05

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA/ITO Suite 400 3701 N. Fairfax Dr. Arlington, VA 22203	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

N/A

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE

No Restrictions

1

13. ABSTRACT (Maximum 200 words)

This is a short course that augments the existing Overview of Megaprogramming course, introducing the concepts, methodologies and application of domain engineering as a strategy for reuse. The presentation covers the concepts of domain analysis and design in a manner suitable for instruction of high school seniors.

The course material contains viewgraphs instructors can use as the basis of lectures. Each viewgraph has accompanying notes that show how to present the viewgraph and suggest topics for discussion. A teachers workbook of background material and exercises is also included.

14. SUBJECT TERMS Software engineering, software reuse, course, Domain Engineering, Domain Analysis, Domain Design, Architectures.	15. NUMBER OF PAGES 170
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

CONTENTS

Tab

This page intentionally left blank.

ACKNOWLEDGMENTS

DISCUSSION - Domain Analysis and Design

There are currently two related courses about which students taking this course should be aware. The first, *Overview of Megaprogramming*, provides an introduction to megaprogramming concepts. The second course, *Megaprogramming With Ada*, provides an introduction to programming using the Ada language with megaprogramming as the underlying methodology. These courses have been taught at high schools in West Virginia and Virginia with positive results.

The purpose of the *Domain Analysis and Design* course is to provide an in-depth look into the domain analysis and domain design phases of domain engineering.

Domain analysis is one of two parallel processes that make up megaprogramming (the other is application engineering). The objective of the first phase of this course is to help students understand the purpose of domain analysis and what domain analysis involves.

The second phase of the course focuses on domain design; students are introduced to the process of creating solutions to the problem area defined during domain analysis.

OBJECTIVES FOR THE COURSE

Students should understand the relationship among megaprogramming, domain analysis, and domain design.

1-1

SOFTWARE PRODUCTIVITY CONSORTIUM

Domain Analysis and Design

Unit 1: Introduction to Domain Analysis

SOFTWARE PRODUCTIVITY CONSORTIUM

1-2

DISCUSSION - Unit 1: Introduction to Domain Analysis

This unit reviews the important concepts from the *Overview of Megaprogramming* course. Megaprogramming as a technique is different from the traditional software development process. Although megaprogramming is still in the research stage, certain concepts of megaprogramming have been used by industry for years.

The traditional approach to software development has many limitations and problems. Megaprogramming addresses many of these limitations and problems by looking at similar projects and solutions, rather than by approaching each project as unique and requiring a unique solution.

This unit highlights the troubles encountered in traditional programming. The basic concepts of megaprogramming are reviewed. The value of megaprogramming is reenforced.

OBJECTIVES FOR THE UNIT

Students are given an introduction to domain analysis and design. The source of the material is presented.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

DISCUSSION - Megaprogramming

To start with, let's review what we learned in the megaprogramming course.

We learned that the motivation for megaprogramming was to help fight the rising cost of developing and maintaining software. We learned that the software lifecycle consists of four stages: recognition of the need for software, development of the software, use of the software, and retirement of the software. We learned that the development process includes requirements, design, coding, testing, delivery, and support. Megaprogramming is the next generation of this development process; it allows software developers to use existing, proven components each time a product is created.

In the megaprogramming course, we learned that almost all software programs (or solutions) address similar problems. Therefore, the programs are similar in many ways. In this course you will learn how the same software programs can be reused to solve new problems that are similar to past problems. Reusing these software solutions is known as software reuse.

Software reuse is the use of an asset (such as code, requirements, documentation, etc.) in more than one system. Reuse may occur within a system, across similar systems, or in widely different systems. For example, this slide illustrates the reuse of obstacle avoidance algorithms. .These algorithms, represented by the white triangle, are extracted from the robot software repository and reused in the development of planetary explorer robot software.

Some of the benefits of software reuse are:
- Productivity improvement
- Increased quality and reliability
- Improved time-to-field systems

STUDENT INTERACTIONS

- Name the four steps in the software life cycle.
- Can you explain how software reuse can lead to the benefits I just mentioned?

OBJECTIVES

- Students review the key principles presented in the megaprogramming course.
- Students should be able to define software reuse and explain some of its benefits.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Software Economics

- **Problem:** Rising cost of software development
- **Solution:** Reuse
  - Use of assets in more than one system
    - Assets = code, requirements, documentation, etc.
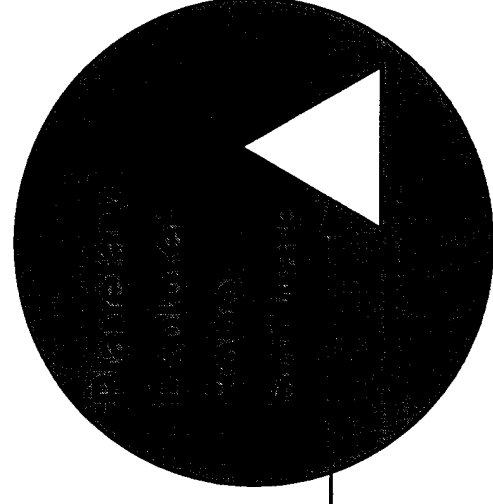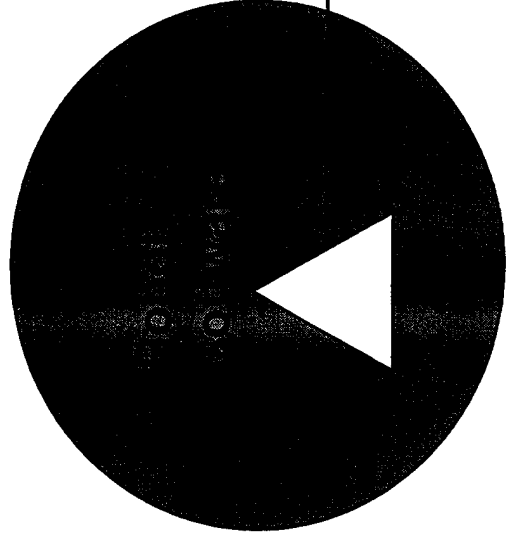
△ = Reuse of obstacle-avoidance algorithms from Robot software repository

1-3

SOFTWARE PRODUCTIVITY CONSORTIUM

DISCUSSION - Domains

In order to implement reuse, an organization must expend time and money to understand and document past problems and their solutions. Since the expense involved can be substantial, the organization must derive considerable benefit from reuse to offset the cost. Reuse of past solutions for present problems works best in situations in which the new problem is similar to past problems. Furthermore, the benefit is greatest when the past problems examined have substantial similarities. Problems that fall into the same domain generally have substantial similarities.

A domain is any group of items, problems, or solutions that have common capabilities or features. Some examples of domains in everyday life are automobiles, video games, clothing, rock music, and country music.

This slide shows an example of the domain of transportation. There are multiple possible physical solutions to the problem of transporting objects by air; a jet passenger plane, a dirigible, and a helicopter are just three. These solutions have important similarities; each moves through air, transports objects, transports people, etc. Note that there are also differences between each solution; one is powered by a jet engine, another is lighter than air, and the third is a rotary aircraft. While similarities must exist for items to be in the same domain, differences reflect how a solution answers a specific variation of the problem. For example, the helicopter would be the answer to the problem of transporting objects stated as: "Objects must be transported by air to places with few or no runways." Differences such as this are important factors in reuse, as we will discuss later.

STUDENT INTERACTIONS

- Can you give some examples of domains?
- For each example you gave, can you name some domains (subdomains) within each domain (e.g., front-wheel-drive cars are a subdomain of automobiles)?

OBJECTIVES

Students should be able to:
- Understand what is a domain
- Explain why reuse emphasizes solving new problems in the same domain as past problems
- Give some examples of similarities and differences in a domain

1-4

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Domains

- Involve problems and solutions that have:
  — Similarities among problems
  — Solutions with common parts
  — Variations among problems and solutions

Domain: Transportation
Problem: Objects must be transported by air

Possible Solutions

1-4
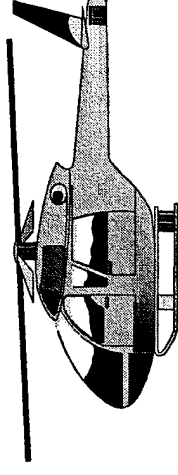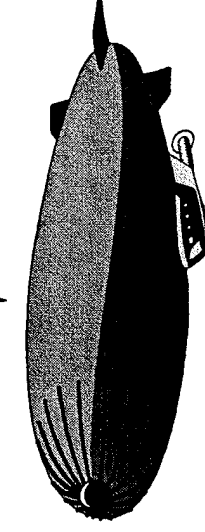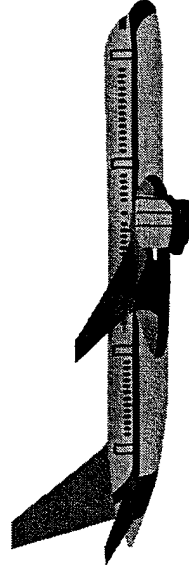
DISCUSSION - Megaprogramming

Megaprogramming is composed of application engineering and domain engineering. Application engineering involves activities related to designing a software solution for a current problem. Domain engineering involves activities that produce products (in the form of knowledge of past problems and solutions) that can be used in application engineering. The relationship of application engineering to domain engineering is shown in this slide. Application engineering is represented by the area above the dotted line, and domain engineering involves the area below the line. They work together in the following manner.

A customer who has a problem contacts an application engineer. The problem is usually vague in nature or poorly stated. In order to solve it, the application engineer must first understand the problem and state it precisely. In traditional software development, the application engineer works from scratch to do this. In megaprogramming, the application engineer can reuse the knowledge of similar past problems derived from domain engineering to state precisely the new problem.

Once the problem is defined precisely, the application engineer must develop a new solution. With megaprogramming, the engineer can reuse the knowledge of past solutions to problems similar to the new problem in order to help design and implement a solution to that new problem. The knowledge of past solutions is derived from domain engineering

The knowledge of past similar problems and of past solutions are both products of domain engineering. The first phase of this course focuses on domain analysis—the domain engineering activity that provides the knowledge of past similar problems. The second phase of the course covers domain design, the domain engineering activity that provides the knowledge of past similar solutions.

STUDENT INTERACTIONS

- How will knowing past problems help to define new problems?
- When would past problems and past solutions be most helpful in solving a current problem? (Answer involves past and present problems being similar, in same domain.)

OBJECTIVES

Students should be able to:

- Understand the difference between application and domain engineering.
- Explain how application and domain engineering are used together to develop a new solution
- Define domain analysis and domain design
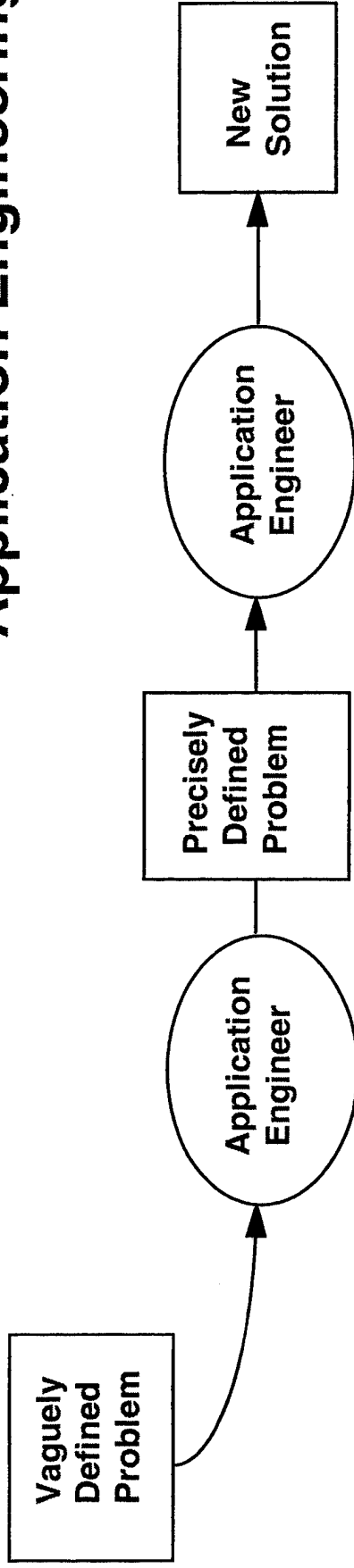
1-5

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Megaprogramming

Application Engineering

Vaguely Defined Problem → Application Engineer → Precisely Defined Problem → Application Engineer → New Solution

Domain Engineering

Knowledge of Past Similar Problems — Domain Analysis

Knowledge of Past Solutions — Domain Design

1-5

DISCUSSION - Domain Engineering

Domain engineering is a process that analyzes an organization's existing software environment in order to derive products that application engineering can use in developing current software solutions. Domain engineering consists of the four major subprocesses or phases that are shown in this slide.

- Domain management develops the domain management plan. This plan defines long-range and near-term domain engineering objectives, and organizes and/or defines domain resources to achieve those objectives.
- Domain analysis involves scoping, identifying, collecting, organizing, analyzing, and representing the relevant information in a domain concerning the problems resolved in that domain.
- Domain design involves developing a generic design for the solutions to problems in the domain.
- Domain implementation is responsible for constructing reusable software products consistent with the results of the domain analysis and domain design processes.

STUDENT INTERACTIONS

- Which of the four domain engineering phases do you think provides knowledge of past problems?
- Which of the four domain engineering phases do you think provides knowledge of past solutions?

OBJECTIVE

Students should be able to define the four phases of domain engineering.

1-6

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Domain Engineering

Domain Management
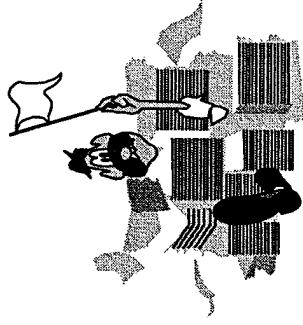
Domain Analysis
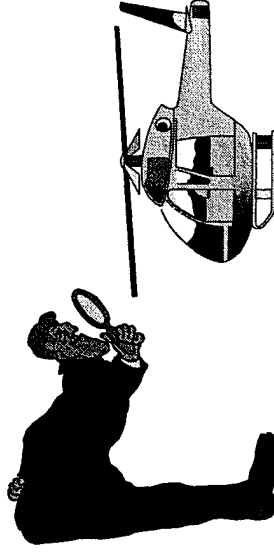
Domain Design

Domain Implementation

SOFTWARE PRODUCTIVITY CONSORTIUM
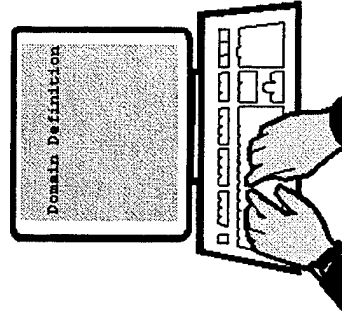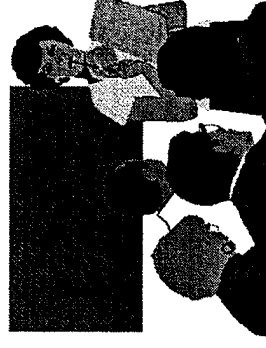
DISCUSSION - Domain Analysis

As I stated before, the application engineer uses the products of domain analysis to gain knowledge of past similar problems. This course concentrates on the two major areas of domain analysis, domain scoping and domain modeling, which are highlighted on this slide.

Domain scoping is the first, and perhaps the most critical, task in the domain engineering process. Domain scoping is identifying one domain from among the many with which a project starts. This domain should be one that appears to have a high potential for return on investment. Once identified, the domain must be precisely defined.

Domain modeling involves gathering detailed information on the domain and then organizing this information. The information is organized into visual representations or models. These models show the knowledge of past problems, which application engineers can then use to define new problems precisely. In this course, domain modeling is presented in three units; these cover scoping individual systems, producing a single model for the domain, and producing a model that attempts to predict new problems in the domain.

OBJECTIVE

Students should be able to define the two major activities of domain analysis and explain what each involves.

1-7

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Domain Analysis

Application Engineering

Vaguely
Defined
Problem → Application Engineer → Precisely Defined Problem → Application Engineer → New Solution

Knowledge of Past Similar Problems

Knowledge of Past Solutions

Domain Design

Domain Engineering

Domain Analysis
- Scoping
- Modeling

Domain Analysis

1-7

UNIT 1: INTRODUCTION TO DOMAIN ANALYSIS

Summary

The purpose of the *Domain Analysis and Design* course is to provide a look at the domain analysis and domain design phases of domain engineering.

Domain analysis is one of two parallel processes that comprise megaprogramming (the other is application engineering). The objective of the first phase of this course is to help students understand the purpose of domain analysis and what domain analysis involves. The second phase of the course focuses on domain design; students are introduced to the process of creating solutions to the problem area defined during domain analysis.

The traditional approach to software development is characterized by many limitations and problems. Megaprogramming addresses limitations and problems by looking at similar projects and solutions, rather than by approaching each project as unique and requiring a unique solution.

The motivation for using megaprogramming was to help fight the rising cost of developing and maintaining software.

The software life cycle consists of four stages:

- Recognition of the need for software

- Development of the software

- Use of the software

- Retirement of the software

The development process includes requirements analysis, design, coding, testing, delivery, and support. Megaprogramming is the next generation of this development process; it allows software developers to use existing, proven components each time a software product is created.

Almost all software programs or solutions address similar problems. Therefore, the programs are similar in many ways. The same software programs (or parts of programs) can be reused to solve new problems that are similar to past problems. Reusing these software solutions is known as software reuse.

Software reuse is the use of an asset (such as code, requirements, documentation, etc.) in more than one system. Reuse may occur within a system, across similar systems, or in widely different systems.

Some of the benefits of software reuse are:

- Productivity improvement

- Increased quality and reliability

- Improved time-to-field systems

In order to implement reuse, an organization must expend time and money to understand and document past problems and their solutions. Since the expense involved can be substantial, the organization must derive considerable benefit from reuse to offset the cost. Reuse of past solutions for present problems works best in situations in which the new problem is similar to past problems. Furthermore, the benefit is greatest when the past problems examined have substantial similarities. Problems that fall into the same domain generally have substantial similarities.

A domain is any group of items, problems, or solutions that have common capabilities or features. Some examples of domains in everyday life are automobiles, video games, clothing, rock music, and country music.

Megaprogramming is composed of application engineering and domain engineering. Application engineering involves activities related to designing a software solution for a current problem. Domain engineering involves activities that produce products (in the form of knowledge of past problems and solutions) that can be used in application engineering.

Once a problem is defined precisely, the application engineer must develop a new solution. With megaprogramming, the engineer can reuse the knowledge of past solutions to problems similar to the new problem in order to help design and implement a solution to that new problem. The knowledge of past solutions is derived from domain engineering.

The knowledge of past similar problems and past solutions are both products of domain engineering. Domain engineering is a process that analyzes an organization's existing software environment in order to derive products that application engineering can use in developing current software solutions. Domain engineering consists of four major subprocesses:

- Domain management develops the domain management plan. This plan defines long-range and near-term domain engineering objectives and organizes/defines domain resources to achieve those objectives.

- Domain analysis involves scoping, identifying, collecting, organizing, analyzing, and representing the relevant information in a domain concerning the problems resolved in that domain.

- Domain design involves developing a generic design for the solutions to problems in the domain.

- Domain implementation is responsible for constructing reusable software products consistent with the results of the domain analysis and domain design processes.

The first phase of this course focuses on domain analysis—the domain engineering activity that provides the knowledge of past similar problems.

The second phase of the course covers domain design, the domain engineering activity that provides the knowledge of past similar solutions.

DISCUSSION - Unit 2: Domain Scoping

Many problems that you have encountered in your schoolwork have seemed insurmountable at first: a difficult math problem, a project, a report on a large topic. Most likely you have approached these complex tasks by breaking them down into a set of smaller tasks and then performing each of these smaller tasks.

Many problems encountered in developing software systems also appear to be insurmountable at first. However, just as you have seen before, the large software problem can be broken down, or decomposed, into a smaller set of problems. The large problem is then solved by solving each of the smaller problems.

That is the basic idea behind what we call domain scoping. There is a bit more behind it, however, which we will explain in this unit.

STUDENT INTERACTIONS

- Can you think of some tasks you have decomposed into smaller tasks?
- What were the smaller tasks?

OBJECTIVES FOR THE UNIT

Students should be able to:

- Explain the need for domain scoping
- Explain the terms domain of interest, domain of focus, and context diagram
- Understand one method for selecting a domain of focus
- Understand the use of a context diagram in scoping the domain of focus
- Understand the need for verification

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Unit 2: Domain Scoping

SOFTWARE
PRODUCTIVITY
CONSORTIUM

2-1

DISCUSSION - Domain Scoping: Why?

Many tasks in building software are large and complex. Sometimes software developers, usually working as a team, will not be able to resolve the entire task in the time allowed and with the resources allocated (the people, hardware, and software). In order to complete the task, the developer will break the task into smaller parts. If performing all the smaller tasks is still too much, the developer (with the customer's agreement) will do one specific part of the development first. This activity of decomposing a large task and choosing a smaller part is known as "scoping the task."

Performing domain engineering begins as a complex task that must be broken down into smaller parts. A decision is usually made to start with one part and then tackle the others.

STUDENT INTERACTIONS

- In choosing a class project, have you ever started with an idea for a very large project and then realized it was too large and chosen a smaller part of that topic?
- Can you give some examples?

OBJECTIVE

Students should be able to understand the need for domain scoping.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Domain Scoping: Why?

- Domain engineering is a complex problem.

- Scoping reduces a complex problem to a reasonable size.
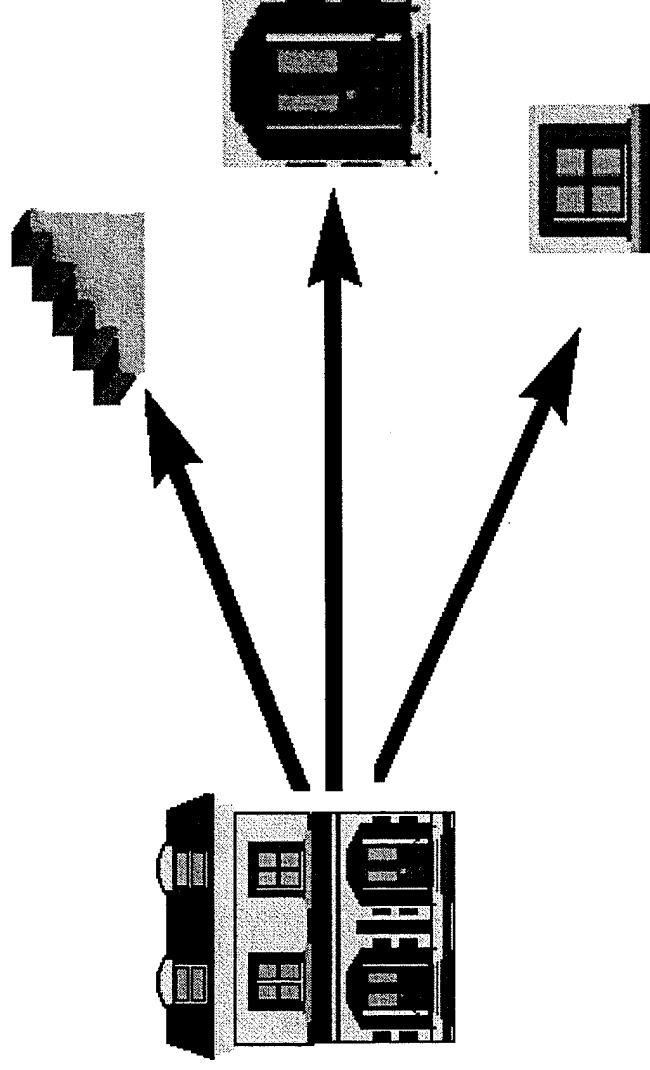
DISCUSSION - Domain Scoping: What?

Often an organization—a business or a government agency, such as NASA or the Department of Defense—has many domains that are candidates for domain engineering. One of the domain engineer's first task, and perhaps the most crucial to ensure success in the domain engineering effort, is to identify one domain that appears to have a high potential for success—a domain for which a domain engineering effort can be completed on schedule and with the resources allocated.

A schedule is simply the time allotted for the DE effort. Resources are the people (domain engineers and others), hardware, software, and any other equipment needed to perform the domain engineering effort.

Besides identifying a single domain, a specific description of this domain must be developed. In this way everyone working on the project will understand what they have agreed to work on.

STUDENT INTERACTION

Before beginning a large project, some teachers demand that you provide a short description of what you intend to work on. Why do they do this?

OBJECTIVE

Students should be able to explain the two basic steps of domain scoping.

Domain Scoping: What?

- Identifies a single domain from a set of potential domains.

- Specifically describes the area within the single domain that will be the focus of the domain engineering effort.

Single Domain

Potential Domains

2-3
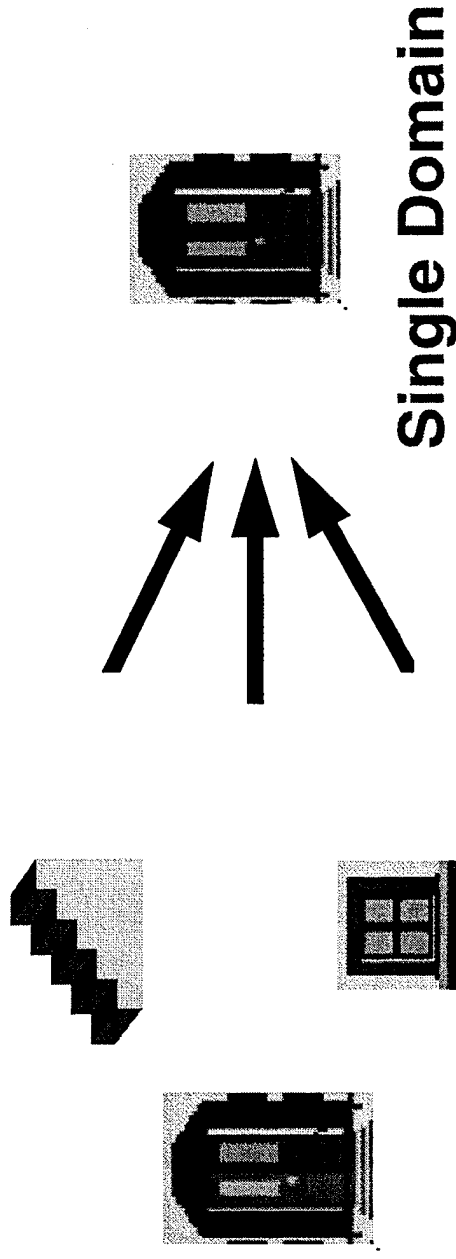
DISCUSSION - Domains of Interest

Domains of interest (DOIs) are the set of potential domains considered for domain engineering. Usually there is a relationship among the functions (or activities) that the DOIs performed. However, the DOIs may be related in some fashion other than function.

Consider this example:

One domain engineering effort considered five domains of interest within NASA: mission operations, shuttle furnaces, flight software, wind tunnels, and shuttle engines.
The only obvious relationship is that they are all in the same organization, NASA.

In the megaprogramming course, we discussed the concept of commonality and showed that commonality between robot architectures could be found by comparing models of the different architectures. Similarly you can find commonality between domains of interest by comparing models of the domains. A number of different modeling approaches can be used. We will use functional models, which are hierarchical models of the functions in the domain.

STUDENT INTERACTIONS

- Suppose you were on a team charged with performing domain engineering for General Motors. Can you think of possible domains of interest?
- What about possible domains of interest in a manufacturer of televisions? of stereos?
- Can you think of anything in common among the various DOIs you identified for General Motors?
- Can you think of anything in common between the DOIs you identified for televisions and for stereos?

OBJECTIVES

Students should be able to:

- Explain what is a domain of interest
- Understand that domain engineering looks for commonalities between the DOIs—often, but not always, by comparing models

2-4

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Domains of Interest

- Domains of interest (DOIs) are the domains considered for analysis.

  — Each of the following could be a DOI: a robot to pick corn, a robot to rescue people, a robot to pick up litter.

- Look for commonalities in problems between DOIs.
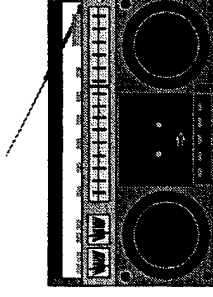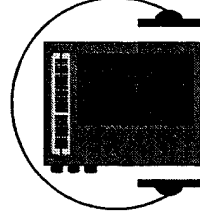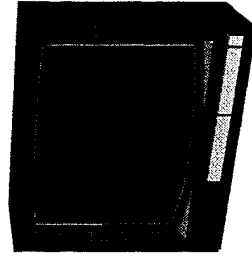
Domains of Interest

DISCUSSION - Requirement and Function

Requirements are statements of what a product must do to solve a problem in the domain. For example, a problem may be: "A vehicle must carry both cargo and people." A requirement may then be stated as: "The vehicle shall be able to carry from one to three people while carrying up to 1 ton of cargo."

Functions are simply the activities that a product must perform to meet the requirements, that is, to solve the problems in a domain. If the analyst is lucky, the specific functions are already documented for the domain. However, it is more likely that the analyst will have to interview experts working in the domain and/or derive functions from requirements in the domain.

The requirement for carrying cargo and people would probably be interpreted as calling for two separate functions: Carry Cargo and Carry People. Consider one of these functions: Carry People. This could be broken down or decomposed into subfunctions such as Provide Space for People, Provide Seating for People, Allow Entrance to People Space, and Allow Exit from People Space.

Functions, especially those that are derived from requirements, can often be decomposed. This decomposition allows us to model the domain as a hierarchical function model or functional model. This decomposition is a visual representation of a problem broken into parts (or functions) that must be included to solve the problem.

STUDENT INTERACTIONS

- Suppose you were able to have someone build a stereo system just for you. What requirements would you give them?
- Can you think of functions that the system would have to perform to meet these requirements?
- Can requirements interact? For example, is it OK to reduce the maximum weight of cargo if there are three crew members rather than one?

OBJECTIVES

Students should be able to:

- Explain what is a requirement
- Explain what is a function
- Propose functions to fit a requirement

2-5

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Requirement and Function

Requirement = statement of what a product must do to solve a problem

Function = activity that fulfills or partially fulfills a requirement

Provide Stereo-Based Audio

Output Sound

SOFTWARE
PRODUCTIVITY
CONSORTIUM

DISCUSSION - Audio DOI: Functional Model

This slide shows an example of a functional model for a requirement to provide stereo-based audio. Obvious physical solutions to this problem bring to mind. However, note that a functional model is a visual description of the problem. You do not specify any solution at this time. Specifying a solution occurs in design. Describing the problem in a nonphysical manner is done during analysis. In that way, designers are freer to be creative and to consider all the constraints on design before making a final choice.

A functional model starts with a few functions that can meet the high-level requirement: Control Volume, Control Remotely, Tune Station, etc. These functions can then be decomposed, based on more detailed requirements or other information. Other information may be found in documentation or by interviewing system developers and users (domain experts).

In this example, the Control Remotely function is decomposed into the functions of Select Volume, Select Media, and Play Selected Media.

STUDENT INTERACTIONS

- Can you think of some functions that would further decompose the increase volume and decrease volume functions? Remember, think in generalities and name activities, not physical parts.
- Are there functions at the top level that you would like to see added to these?

OBJECTIVES

Students should be able to understand that analysis involves defining a problem precisely without specifying a particular solution.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Audio DOI: Functional Model

Provide Stereo-Based Audio

- Control Volume
  - Increase Volume
  - Decrease Volume
- Control Remotely
  - Select Volume
  - Select Media
  - Play Selected Media
- Tune Station
- Output Sound
  - Output Normal Sound
  - Output Auxiliary Sound
- Provide Media Type Choice
  - Choose CD Player
  - Choose Tape Player
  - Choose AM/FM Tuner

2-6
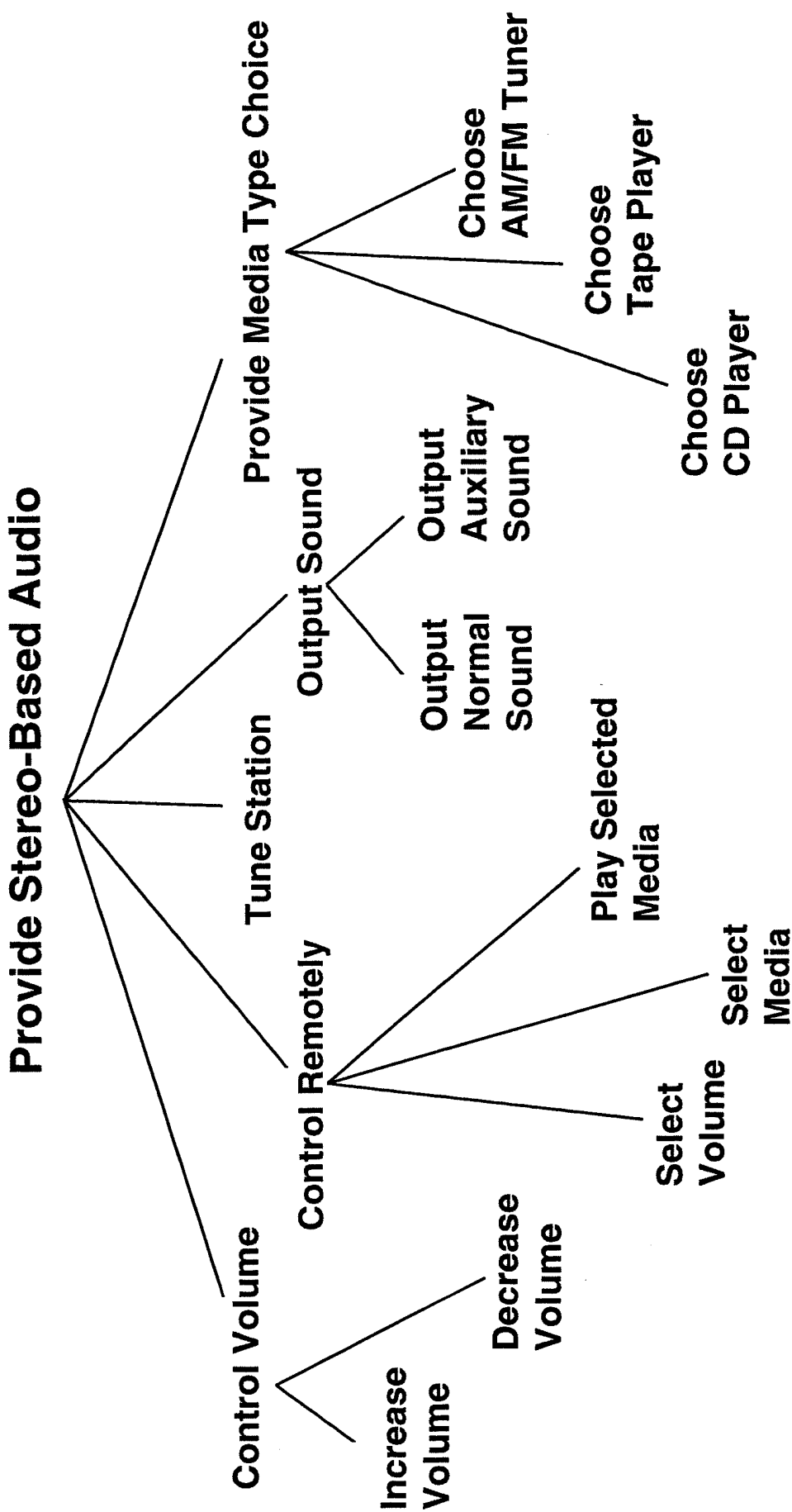
DISCUSSION - Video DOI: Functional Model

This slide shows an example of a functional model for a requirement to provide video. Again, physical solutions to this problem spring to mind. However, note that if you specify television, you ignore the alternatives of cinema and computer-based video. Stopping at a description of the problem allows these and possible novel alternatives to be considered.

STUDENT INTERACTION

Note that even when you try to stay at a problem-description level, the physical often "creeps in." For example, the subfunctions for the function Provide Media Type Choice reflect current physical possibilities. Can you think of others not listed here? Try to think of novel possibilities.

OBJECTIVE

Students should be able to understand that past solutions can bias the analysis.

2-7

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Video DOI: Functional Model

Provide Video

Provide Media Type Choice
- Choose VCR
- Choose Laserdisc
- Choose Stereo

Control Remotely
- Select Channel
  - Increment Channel
  - Decrement Channel
- Play Selected Media
  - Select Media
- Select Volume
- Select Channel

Control Volume
- Increase Volume
- Decrease Volume

SOFTWARE PRODUCTIVITY CONSORTIUM
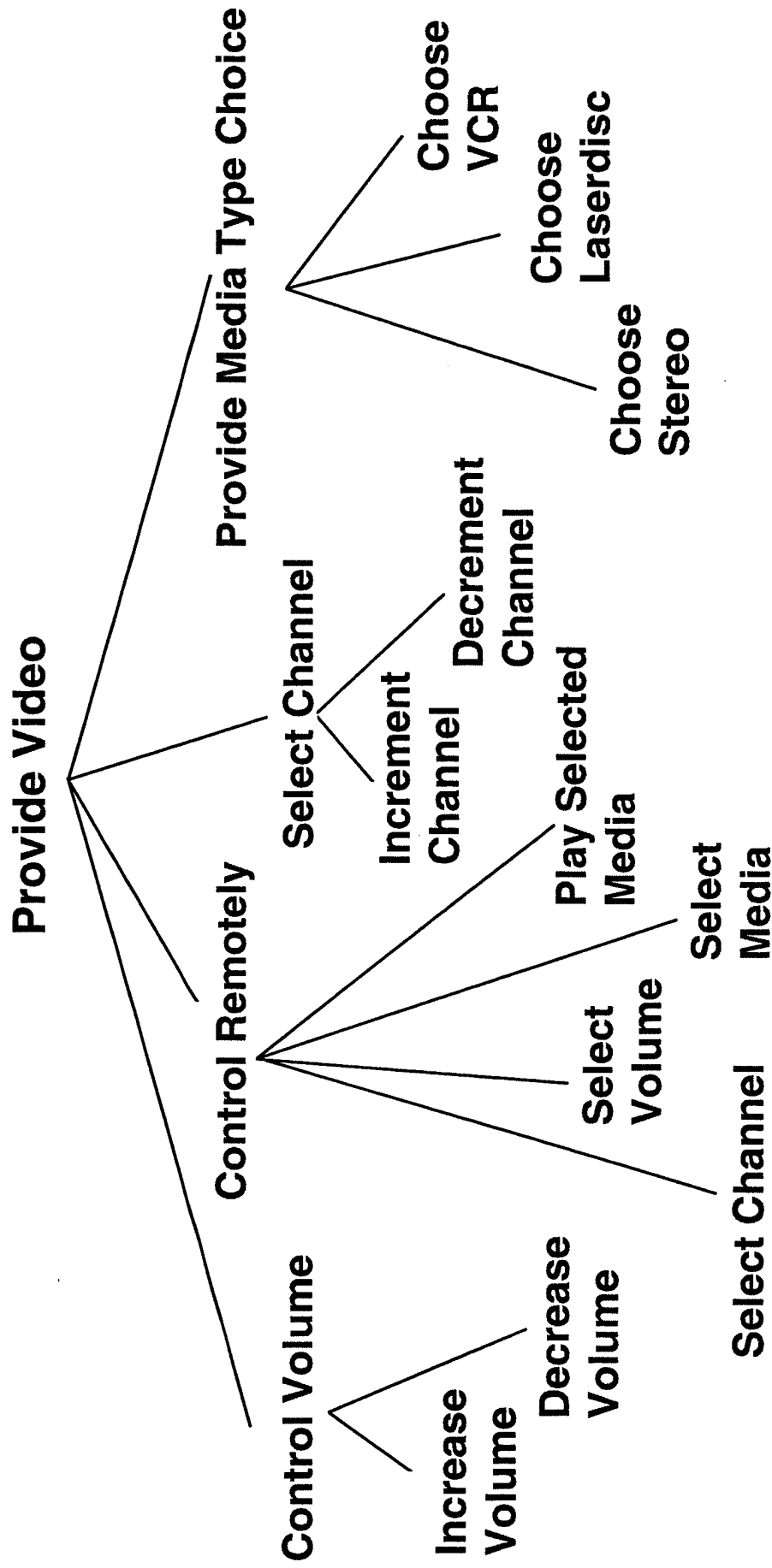
DISCUSSION - Select Domain of Focus

Once you have developed your models of the domains of interest (DOIs), you must choose a domain of focus (or DOF). Actually more than one DOF can be chosen and a priority set as to which one to work on first, second, and so on. However, we will stick with choosing one DOF in this course.

The simplest way to choose a DOF is to chose one of the DOIs. Sometimes this is done by management decision (you are told which DOI to chose) and sometimes by using certain selection criteria. In many problem-solving activities, we use a certain set of criteria to make a decision. In choosing a car we may consider price, performance, gas mileage, handling, reliability, opinions of friends, etc. Similarly, in choosing a DOF you may use selection criteria. (The teacher may discuss the selection criteria if there is sufficient time. They are described in the Teachers Notes for Exercises in the course workbook.)

The other way to choose a DOF is related to the megaprogramming concept of finding commonalities in problems, that is, to select a domain that has some commonality across two or more problem descriptions. We will use this approach to look for commonality between the problem descriptions, the functional models, we have developed. We do this by identifying the functions that the two models have in common.

STUDENT INTERACTION

Suppose you were given the DOIs of refrigerators, stoves, and microwaves. Name some areas of commonality between them. Choose one of these areas and state why you chose it.

OBJECTIVES

Students should be able to:

- State the two major ways to choose a DOF
- State two ways to choose a single DOI as the DOF

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Select Domain of Focus

- Domain of focus (DOF):

  — May be one of the DOIs, or

  — Is a domain that has some commonality across two or more DOIs

- Common functions show commonality between DOIs.

- If more than one function in common, then select one DOF based on:
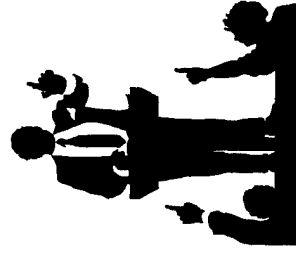
Management Decision

Cost   Documentation

Reuse potential

Domain experts available

Selection Criteria

2-8

DISCUSSION - Common Functions

Here we show the commonalities in our descriptions of the audio and video problems. There are two functions in common: Control Volume and Control Remotely. Control Volume is a commonality since the decomposition of this function is the same for both domains of interest. The same is true of Control Remotely.

You may also have noticed that the function Provide Media Type Choice appears in both DOIs. However, the decomposition of the function is different for each domain. Thus, this function is not a commonality between the two domains. This does, however, represent the important concept of differences or variability between domains. We will discuss this topic in more detail later.

We will assume that management has made the decision to chose Control Remotely as the DOF.

STUDENT INTERACTIONS

On the models of audio DOI and video DOI, can you think of ways to express the decomposition of the Provide Media Type Choice function that would have led to this function being a commonality? Do you think this is a better or worse way to represent this function? Defend you choice.

OBJECTIVES

Students should be able to:

- Understand why the two functions chosen describe a commonality of the problem
- Understand why the Provide Media Type Choice function is not a commonality as the problem has been described

2-9

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Common Functions

Common Functions

Common Functions
├── Control Volume
│   ├── Increase Volume
│   └── Decrease Volume
└── Control Remotely
    ├── Play Selected Media
    ├── Select Media
    └── Select Volume
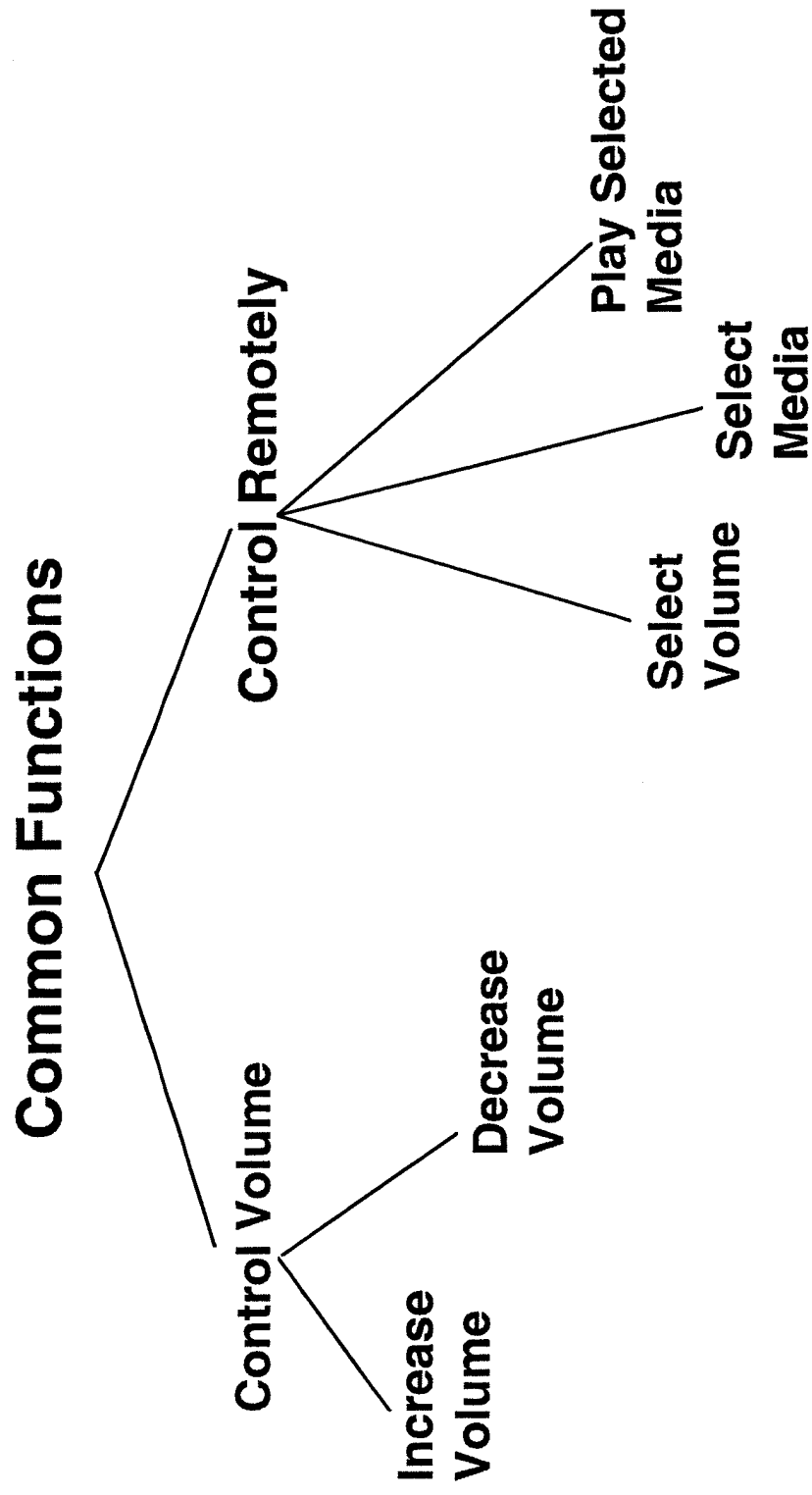
DISCUSSION - Scope DOF

The scoping process in domain analysis is iterative. First we scope the DOIs to find the DOF, and then we scope the DOF to define precisely the problem area. Scoping the DOF ensures that the problem area to be addressed is clearly specified and understood by all involved. In describing a problem, it is important to describe not only what is within the problem area, but what is outside it. This clear definition of the DOF forms the basis for communication about the purpose of the domain engineering effort.

The key to this definition is the context diagram, a simple high-level visual description that is basically another type of model. The domain of focus is shown as central, usually within a circle. The systems, personnel, and/or organizations that interact with the DOF are shown outside the DOF, usually as boxes. These are referred to as the interfaces to the DOF. Arrows between the interfaces and the DOF show the inputs and outputs to the DOF. Inputs and outputs between the boxes are not shown. Since the DOF is the problem area of concern, inputs and output that do not relate directly to the DOF are not of interest.

The context diagram should be developed from a functional perspective. However, what usually occurs is that there are existing systems, and a physical name is chosen from these systems for the DOF. For example, instead of Control Remotely you are likely to see Remote Controller—a physical, solution-oriented term. This does not imply that we no longer continue to describe the problem in domain analysis; we still do. We are just pointing out a seeming inconsistency that occurs in real-life domain analysis.

STUDENT INTERACTION

Have you ever thought you made an agreement with someone, only to find out later that the two of you agreed on different things? In other words, you each understood the agreement differently. Do you think it would have helped to specify what you had **not** agreed upon? Can anyone describe such an experience?

OBJECTIVES

Students should be able to:

* Explain the components of a context diagram
* Understand why it is important to scope the DOF, i.e., to specify the problem area

2-10

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Scope DOF

- Context diagram clearly defines DOF interfaces

Interface to DOF

Input to DOF

Output from DOF

DOF

Input to DOF

Output from DOF

Output from DOF

Input to DOF

Interface to DOF

Input to DOF

Output from DOF

Interface to DOF

SOFTWARE PRODUCTIVITY CONSORTIUM
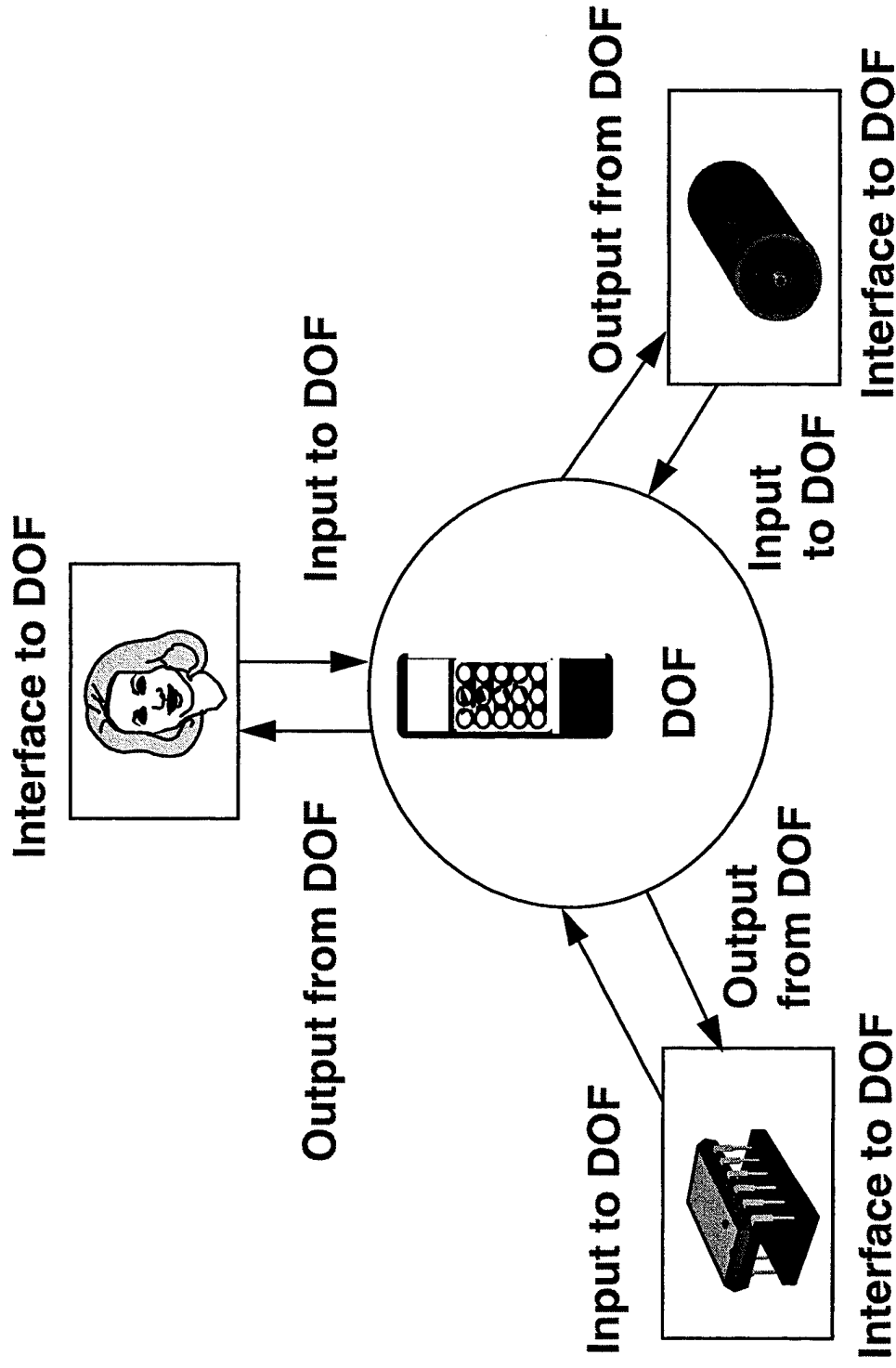
DISCUSSION - Context Diagram for Control Remotely Domain

The domain of focus is shown here as the central circle—Control Remotely. The boxes show the key objects, people, systems, etc., that are interfaces to the domain of focus. For this DOF, any person can interact with the Control Remotely function. This interaction consists of inputting various Select Functions and viewing a display (an output from Control Remotely).

Scoping of the DOF should include a context diagram and a verbal description. An appropriate verbal description would be:

> The Control Remotely function sends electrical Signals to both the Control Board and the Infrared Sensor. A Power source sends a DC Supply of power to the Control Remotely function.

This verbal description can be enhanced by describing some other characteristics, such as typical systems in which this DOF is used.

STUDENT INTERACTION

- Teacher: Lead students in drawing a context diagram for the domain of focus of Control Volume.

OBJECTIVE

Students should be able to provide a verbal description of the domain of focus.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Context Diagram for Control Remotely Domain

Power

Infrared Sensor

DC Supply

Signals

Control Remotely

Display

Signals

Person

Select Functions

Control Board

2-11

SOFTWARE PRODUCTIVITY CONSORTIUM
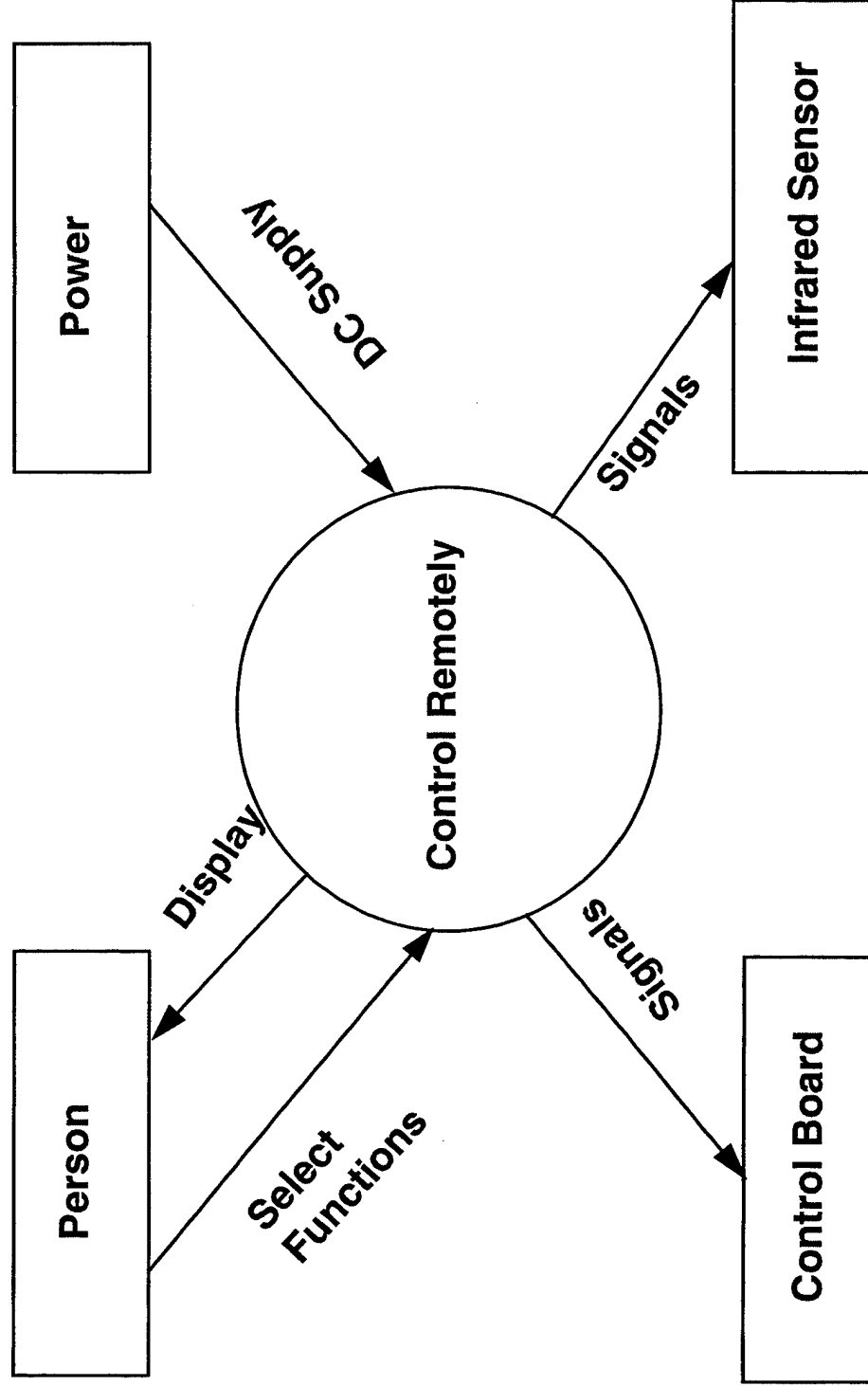
DISCUSSION - Validattion

In any development endeavor, you must ensure that the work you are doing meets the customer's needs. It is safest to pursue an iterative development procedure. This involves producing a small part of the final product and getting feedback from the customer on each part produced.

This has a number of advantages. First, it ensures that each part meets the customer needs and increases the probability that the final product will meet those needs. Second, once a part is validated, the developers can be confident they are building on a sound foundation; any second-guessing or worrying about "completed work" is minimized. Third, the customer stays involved. Requirements often change during the development of a software project. If the customer is continually involved, the changing requirements can be introduced early and minimize their impact. Drastic requirement changes introduced near the end of a project have led to cost overruns and the failure of many projects.

During domain analysis, you must check with the customer **at least** at the completion of each major product.

Following the scoping of the DOF, meet with the customer and:

- Ask for **and expect** corrections to the context diagram and your textual description.
- Ask the customer if a domain engineering effort on this DOF will provide a beneficial service. If not, ask for suggestions on modifying the DOF or choosing a different DOF.

STUDENT INTERACTION

Suppose a company was building a remote controller based on your analysis. Furthermore, suppose the requirements had changed, but you did not find out about this until the company was ready to start producing remote controllers. What could the effect be? What steps could have been taken to minimize this effect?

OBJECTIVE

Students should be able to understand the importance of validation.

2-12

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Validation

- Ensure that the work you are doing meets customer needs.

- Never wait until development is complete to ask the customer if you are meeting those needs. To do so practically ensures failure!

- Check with the customer at least at the completion of each major product.

2-12

SOFTWARE PRODUCTIVITY CONSORTIUM
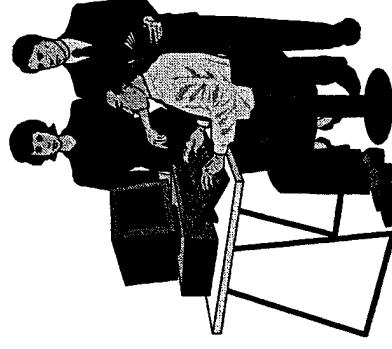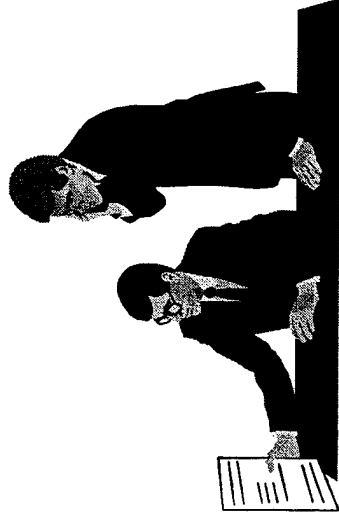
DISCUSSION - Domain Scoping: Summary

Domain scoping is just the application of an activity you have used in the past: finding a small, manageable problem within a large, complex problem.

We have discussed how domain scoping can be accomplished. First, model the set of possible domains (the domains of interest). Second, find commonalities between the models. One way is to look for common functions. Third, select a small, manageable problem (the domain of focus). This can be done by management decision or by selection criteria if there is more than one common area.

Another model, the context diagram, is then produced to describe the domain of focus graphically. A textual description should accompany this graphic.

Finally, the results, especially the context diagram, should be verified with the customer.

STUDENT INTERACTIONS

Review:

- What does domain of interest refer to?
- What does domain of focus refer to?
- What is a requirement?
- What is a function?
- Why do we try to find commonalities?
- Why is a context diagram necessary?
- What are the parts of a context diagram?
- State two reasons for performing validation.

OBJECTIVES

Students should be able to:

- State the major activities in domain scoping
- Define the terms reviewed above

2-13

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Domain Scoping: Summary

- Scoping is needed to:

  - Reduce large, unmanageable set of potential needs, options and alternatives to manageable size

  - Define the domain and potential products precisely

- Functional models for DOIs can be derived from requirements.

- The DOF could be one of the common functions.

- A context diagram describes the interfaces of the DOF.

- Validation increases the probability of success.

2-13

UNIT 2: DOMAIN SCOPING

Summary

Many problems encountered in developing software systems appear to be insurmountable at first. However, a large software problem can be broken down or decomposed into a smaller set of problems. The large problem is then solved by solving each of the smaller problems.

Likewise, domain engineering begins as a complex task that must be broken down into smaller parts. A decision is usually made to start with one part and then tackle the others. This activity of decomposing a large task and choosing a smaller part is known as "scoping the task."

This is the basic idea behind what we call domain scoping.

Often an organization has many domains that are candidates for domain engineering. One of the domain engineer's first task, and perhaps the most crucial to ensure success, is to identify one domain that appears to have a high potential for success—a domain for which the domain engineering effort can be completed on schedule and with the resources allocated. Besides identifying a single domain, a specific description of this domain must be developed. In this way, everyone working on the project will understand what they have agreed to work on.

Domains of interest (DOIs) are the set of potential domains considered for domain engineering. Usually there is a relationship among the functions (or activities) that the DOIs perform. However, the DOIs may be related in some fashion other than function.

Requirements are statements of what a product must do to solve a problem in the domain. Functions are simply the activities that a product must perform to meet the requirements, i.e., to solve the problems in a domain.

Functions, especially those derived from requirements, can often be decomposed. This decomposition allows us to model the domain as a hierarchical function model or functional model. This decomposition is a visual representation of a problem broken into parts (or functions) that must be included to solve the problem.

Once you have developed your models of the domains of interest (DOIs), you must choose a domain of focus (DOF). The simplest way to choose a DOF is to select one of the DOIs. Sometimes this is done by a management decision and sometimes by using certain selection criteria to make a choice. The other way to choose a DOF is related to the megaprogramming concept of finding commonalities in problems, i.e., to select a domain that has some commonality across two or more problem descriptions.

The scoping process in domain analysis is iterative. First we scope the DOIs to find the DOF, and then we scope the DOF to define precisely the problem area. Scoping the DOF ensures that the problem area to be addressed is clearly specified and understood

by all involved. In describing a problem, it is important to describe not only what is within the problem area, but what is outside it. This clear definition of the DOF forms the basis for communication about the purpose of the domain engineering effort.

The key to this definition is the context diagram—a simple, high-level visual description that is basically another type of model. The context diagram should still retain a functional perspective. However, what usually occurs is that there are existing systems, and a physical name is chosen from these systems for the DOF.

Scoping of the DOF should include a context diagram and a verbal description. This description can be enhanced by describing some other characteristics, such as typical systems in which this DOF is used.

In any development endeavor, you must ensure that the work you are doing meets the customer's needs. It is safest to pursue an iterative development procedure. This involves producing a small part of the final product and getting feedback from the customer on each part produced. During domain analysis, you must check with the customer **at least** at the completion of each major product.

Domain scoping is just the application of an activity you have used in the past: finding a small, manageable problem within a large, complex problem.

Exercises

Exercise 1

In the area of transportation, list some domains that support transportation.

Exercise 2

List some common functions of the domains you described in Exercise 1. Develop a method for choosing between common functions, and then choose a function.

Exercise 3

Boom Industries (BI) has won a contract from the Space Agency to construct a Planetary Explorer Robot (PEBOT) that will be used to explore, gather materials, test materials, and store and transmit data back to earth. The PEBOT must have the capability to sustain itself in an environmentally hostile and inadequately mapped region of any planet. A further constraint on BI is to develop the PEBOT from existing materials due to budget and time constraints.

BI has identified the need to analyze various types of mobility systems in order to choose the appropriate type of mobility system for the PEBOT. BI produces robots that are stationary, such as automotive manufacturing robots. To support the requirement of navigating unknown terrain, BI must consider contracting the work of designing and implementing the software and hardware aspects of the mobility system for the PEBOT.

After researching other companies that construct mobile robots, BI has identified United Robot Workers, Inc. (URW) as a company that has gained a solid reputation for designing and mass producing mobile robots for a number of commercial missions. Given URW's extensive experience in robot design, BI decided to team with them to design the PEBOT.

URW engineers realized that PEBOT had to have some unique characteristics to adapt to an environment that essentially is unknown. However, based on URW's experience constructing mobile robots in the domains of agriculture, search and rescue, and sanitation, they noticed similarities in some of the PEBOT's mobility requirements with the robots they had previously designed. Naturally they decided to reuse as much of the existing mobility systems data and design as possible. What they lacked was a method by which to choose the appropriate reusable items from the existing robot mobility systems.

BI and URW engineers, now known as the PEBOT Team, researched various software development methodologies for one that supported the concepts of reuse. After an exhaustive search, they decided to try a new, emerging software development method, known as domain engineering, to assist them in designing the PEBOT. They discovered that the domain engineering discipline exists to support software reusability. Domain engineering accomplishes this by identifying the common and different requirements, functions and designs within a given domain. The common elements are

what become the reusable assets. The PEBOT Team realized that domain engineering could assist them in identifying the reusable pieces among their current selection of robots mobility systems. They hope that once the common aspects are identified, they can map them to the PEBOT.

The PEBOT Team has two major objectives:

- Analyze existing mobility systems to identify reusable items

- Design and produce the PEBOT

The first thing that the engineers did was to gather all the information about their existing mobility systems designs and organize it into the categories of customer needs, requirements documents, design documents, code, customer feedback, etc. Concurrently they began to gather information on the PEBOT and the new domain of planetary exploration. The sheets that follow are the requirements specification sheets for the existing robots and their mobility systems.

You will use the information in the requirements specifications sheets to perform the following activities:

Activity 1

Carefully identify and analyze all requirements and derive the functions for each robotic mobility system. Your job, as part of the PEBOT Team, is to apply the principles you learn in class to arrive at the common and different functions of the mobility systems for each robot. Using these requirements, define the various functions necessary to fulfill those requirements.

Activity 2

From the information in the requirements specifications sheets, create high-level models for the mobility systems in each robot. Represent the information in generic, high-level stages and decompose them accordingly.

Once you decompose the model to a certain level, identify a domain of focus (DOF) that is common among all mobility systems. For simplicity, your DOF is identified: Move_Robot_Body_Parts.

Using the DOF, create the context diagram. Use the requirements specifications sheets for each robot mobility system to identify the various entities with which the DOF must interact. Identify the information flow to and from the entities.

Teacher's Notes for Exercises

The following are notes for the teacher to apply to each exercise presented in Unit 2 of the *Domain Analysis and Design* course. The lists requested of the students need not be comprehensive; they should be adequate to develop their understanding of the material.

Several of the exercises deal primarily with common, everyday hardware and not the software aspects one might expect from this course. The exercises were purposely developed around physical objects to convey the key concepts of the domain analysis methodology more easily to students than would be possible with abstract software concepts.

Exercise 1

In the area of transportation, list some domains that support transportation.

Some responses to note are automobiles, aircraft, and ships. More general responses, which one might expect if an actual model were to be created, are wheeled and nonwheeled.

Exercise 2

List some common functions of the domains you described in Exercise 1. Develop a method for choosing between common functions, and then choose a function.

Functions within an automobile domain: Carry People, Carry Cargo, Provide Motion, Steering, Supply Power

Functions within an aircraft domain: Carry People, Carry Cargo, Provide Motion, Steering, Supply Power

Functions within a ship domain: Carry People, Carry Cargo, Provide Motion, Steering, Supply Power

All of the above are common functions that all three subdomains within the domain of transportation must provide. There could be others, but these few allow for good discussion.

Exercise 3

This is the robot exercise. The graphics that follow present the information the students should generate. Guide them as much as possible towards this end. The documentation provided for their use, called the Requirements Specifications Sheets, identify requirements for the students to analyze in order to derive the supporting functions.

Model 1

Functional Model: Mobility Function of the Agricultural Robot

Model 2

Functional Model: Mobility Function of the Search and Rescue Robot

Search_Rescue_Humans_Mobility

- Move_Robot_Body_Parts
  - Move_Entire_Body
    - Rotate_Body
    - Extend_Legs_Independently
    - Roll_Body_Tracks
  - Move_Fingers
  - Move_Arms
- Provide_Power
  - Apply_Joint_Power
  - Apply_Motion_Power
- Plan_Move_Path
  - Use_Laser_Sensors
  - Create_Adaptive_Path
- Load_Human
  - Pickup_Human
  - Place_Human_Storage
- Unload_Human
  - Rotate_Storage_Container
  - Open_Storage_Door

Model 3

Functional Model: Mobility Function of the Sanitation Robot

Gather_Litter_Mobility

- Move_Robot_Body_Parts
  - Move_Entire_Body
    - Rotate_Body
    - Roll_Body_Tracks
  - Move_Arms
  - Move_Fingers
- Provide_Power
  - Apply_Joint_Power
  - Apply_Motion_Power
- Plan_Move_Path
  - Follow_Programmed_Path
  - Use_Laser_Sensors
  - Adjust_Programmed_Path
- Load_Litter
  - Pickup_Litter
  - Place_Litter_Storage
- Unload_Litter
  - Rotate_Storage_Container
  - Open_Storage_Door

Context Diagram

```
┌─────────────────────────────────────────────────────────────────────┐
│                                                                       │
│   ┌─────────────────────┐                    ┌──────────────────┐     │
│   │ Supervisory System  │ ◀───┐        ┌────▶ │     Sensors      │     │
│   └─────────────────────┘     │        │      └──────────────────┘     │
│                          ╭───────────────╮                            │
│                          │  Move_Robot_  │                            │
│                          │  Body_Parts   │                            │
│   ┌─────────────────────┐ ╰───────────────╯ ┌────────────────────────┐│
│   │ Power_Allocation_   │ ◀──┘        └───▶ │Individual_Joint_Interface││
│   │ Unit                │                    └────────────────────────┘│
│   └─────────────────────┘                                              │
│                                                                       │
└─────────────────────────────────────────────────────────────────────┘
```

Requirements Section

Requirements Specifications Sheet: Agricultural Robot Mobility

This requirements sheet presents the requirements and functional descriptions of the mobility system for a robot that is designed strictly for agricultural needs.

Customer Need

A farmer owns a large corn field and has trouble finding time to harvest it. The farmer wants to know if URW can provide a robot that will harvest the corn without human intervention.

Mission Statement

URW will develop a robot that will harvest corn independent of human direction. The robot must begin its work at a point (origin) and return to that point after completion. The robot must be designed and implemented within a cost limit of $13,500.
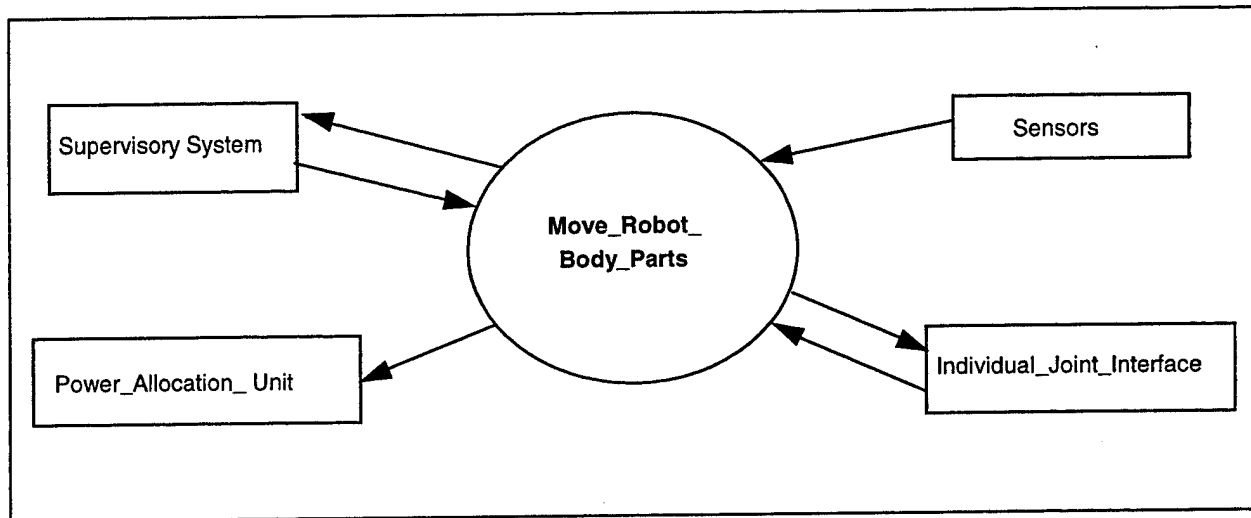
List of Requirements

REQ1:   The robot will harvest between 50 and 500 ears of corn.

REQ2:   The robot will begin its mission at a specified origin and will return to that origin upon mission completion.

Functional Specifications to Meet Requirements

The main requirement is that the robot be able to move so it can gather an agricultural product.

The robot's various body parts will be able to move.

   The robot's entire body will be able to move.

      The robot will be able to rotate its body.

         The robot will rotate 360 degrees right.

         The robot will rotate 360 degrees left.

      The robot will be able to move itself from one point to another point on four wheels.

         The wheels will be able to speed forward.

         The wheels will be able to speed in reverse.

      The wheels will be attached to legs, or suspension stems, that can independently change position.

The robot will have the ability to raise and lower the front of the body.

The robot will have the ability to raise and lower the rear of the body.

The robot will have the ability to gather objects by moving its fingers.

The robot's fingers will be able to open.

The robot's fingers will be able to close.

The robot will gather objects by moving its two-piece single arm.

The two-piece single arm will rotate upwards 180 degrees.

The two-piece single arm will rotate downwards 180 degrees.

The robot will provide power to all its movable systems and to other systems as well.

The robot will apply power from the main source to all joints.

The robot will apply power from the main source to supply motion.

The robot will be able to move itself along a planned path.

The robot will have a mobility unit that can be programmed to follow a particular path.

The robot will know to follow a movement pattern, turn, and return using laser sensors placed along the path.

The robot will take the object and load it into storage.

The robot will pick up a product and rotate its arms.

At the end of rotation, the robot will place the product in a storage container.

Once the robot has completed its mission, it will return to the point of origin and unload its gathered product.

The robot will unload the product by rotating the storage container.

Once the container is rotated to a specified position, it will open the storage door.

Requirements Specifications Sheet: Search and Rescue Robot

This requirements sheet presents the requirements and functional descriptions of the mobility system for a robot that is designed strictly for searching, identifying, and rescuing lost or stranded people.

Customer Need

Over the past two years, the Alaskan Army National Guard (AARNG) has rescued more than 50 people lost in the vast and sometimes uncharted tundra in and around Mount Denali National Forest. The time it takes to mount a rescue party could be the time in which people die, so the AARNG would like to send robots into the tundra to search and locate lost people.

Mission Statement

URW will develop a robot that searches, locates, and rescues, if possible, lost people. The robot will be given the last known coordinates of the search object, and it will initiate its search pattern from that location. Once it locates the object of the search, it will immediately notify AARNG headquarters of its exact location and transmit a beacon signal.

List of Requirements

REQ1:   The robot will search for an object by following a path based on a route stored in the robot, but will adapt itself to a new route.

REQ2:   At the completion of its mission, the robot will (1) stay at its current location, (2) return to its point of origin, or (3) go to a new location.

Functional Specifications to Meet Requirements

The main requirement is that the robot be able to move so it can search, identify, and rescue lost people.

The robot's various body parts will be able to move.

> The robot's entire body will be able to move.

>> The robot will be able to rotate its body.

>>> The robot will rotate 360 degrees right.

>>> The robot will rotate 360 degrees left.

>> The robot will be able to move itself from one point to another point on tracks.

>>> The tracks will be able to roll right.

>>> The wheels will be able to roll left.

The tracks will be attached to legs, or suspension stems that can independently change position.

The robot will have the ability to raise and lower either portion of the front of the body.

The robot will have the ability to raise and lower either portion of the rear of the body.

The robot will have the ability to locate, pick up, or drop off objects by moving its fingers.

The robot's fingers will be able to open.

The robot's fingers will be able to close.

The robot will act upon objects by moving its four independent arms.

Any of the four arms will rotate upwards 180 degrees.

The four arms will rotate downwards 180 degrees.

The four arms will have the ability to extend themselves to probe for life signs.

The four arms will have the ability to retract themselves into the probe container.

The robot will provide power to all its movable systems and to other systems as well.

The robot will apply power from the main source to all joints.

The robot will apply power from the main source to supply motion.

The robot will be able to move itself along a planned path.

The robot will have a mobility unit that can create an adaptive path based on information provided by all the motion sensors.

The robot will know to follow a movement pattern, turn, and react to obstacles with a total of 30 laser sensors.

The robot will take the object and load it into storage.

The robot will pick up an object by using its independent arm motion.

The robot will place the object in a storage container, if told to do so after transmittal of the object's condition.

Once the robot has completed its mission, it will stay at its current location, return to the point of origin, or search for a new location.

The robot will unload the object by rotating the storage container.

Once the container is rotated to a specified position, it will open the storage door so the object can be extracted.

Requirements Specifications Sheet: Sanitation Robot

This requirements sheet presents the requirements and functional descriptions of the mobility system for a robot that is designed strictly for sanitation needs.

Customer Need

The National Park Service is concerned about growing amounts of litter in the national parks. They need a robot that can gather the litter.

Mission Statement

URW will develop a robot that will gather the litter in a prescribed national park independent of human direction. The robot will be given a terrain map of a given park to follow; however, obstacles will arise, and it must be able to avoid the obstacle and return to its original path.

List of Requirements

REQ1:   The robot will gather litter in a prescribed region.

REQ2:   The robot will begin its mission at a specified origin and will return to that origin upon mission completion.

Functional Specifications to Meet Requirements

The main requirement is that the robot be able to move so it can gather litter over a prescribed terrain.

The robot's various body parts will be able to move.

   The robot's entire body will be able to move.

      The robot will be able to rotate its body.

         The robot will rotate 360 degrees right.

         The robot will rotate 360 degrees left.

      The robot will be able to move itself from one point to another point on tracks.

         The tracks will be able to roll right.

         The tracks will be able to roll left.

      The robot will have the ability to gather objects by moving its fingers.

         The robot's fingers will be able to open.

         The robot's fingers will be able to close.

      The robot will act upon objects by moving its two independent arms.

Either of the two arms will rotate upwards 180 degrees.

The two arms will rotate downwards 180 degrees.

The two arms will have the ability to extend themselves to gather an object.

The two arms will have the ability to retract themselves to place the object in a container.

The robot will provide power to all its movable systems and to other systems as well.

The robot will apply power from the main source to all joints.

The robot will apply power from the main source to supply motion.

The robot will be able to move itself along a planned path.

The robot will have a mobility unit that can be programmed to follow a particular path.

The robot will also be able to adjust the programmed path to avoid obstacles and then return to its original path using laser sensors.

The robot will take the object and load it into storage.

The robot will pick up an object and rotate its arms.

At the end of rotation, the robot will place the object in a storage container.

Once the robot has completed its mission, it will return to the point of origin and unload all gathered objects.

The robot will unload the product by rotating the storage container.

Once the container is rotated to a specified position it will open the storage door.

Requirements Specifications Sheet: Planetary Exploration Robot (PEBOT)

Customer Need

The Space Agency needs robots to satisfy the planned requirements for exploring planetary surfaces. Robots will explore potential landing sites and areas of scientific interest, deploy science instruments, and gather samples for analysis. Robots may be required to return to earth. The robots required for such operations will require high level of local autonomy, including the ability to perform local navigation, identify areas of potential scientific interest, regulate on-board resources, and schedule activities, all with limited ground-command intervention.

Mission Statement

URW will develop a robot that will land on any planetary surface, gather data, and autonomously navigate independent of human direction. The robot will be given a prescribed initial path; however, obstacles will arise, and it must be able to avoid the obstacle and independently arrange a new path.

List of Requirements

- Affordable missions

- Reduction in landed mass of 5 kg or less

- Short- to mid-range traverses

- Active and passive power management

- Confirm and map science targets

- Retrieve new rock and other samples

- Accurate deployment and pointing of instruments

- Robot with common sense

DISCUSSION - Unit 3: Domain Modeling, Step 1: Individual System Models

In Unit 2, we discussed domain scoping, the initial and most crucial aspect of domain engineering. Once an organization has identified the initial problem area upon which they wish to focus, they must analyze the problem area—the domain—in detail.

This unit takes you into the second major activity of domain analysis, domain modeling. In domain analysis, the detailed analysis—finding both the common and different aspects of the domain—is accomplished and represented graphically by models.

OBJECTIVES FOR THE UNIT

Students should be able to:

- Explain why exploring previous problems is important to a successful reuse effort
- Define key questions that must be asked in order to develop the models
- Explain why choosing a set of problems is important
- Model the information for each system

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Unit 3: Domain Modeling
Step 1: Individual System Models

3-1

DISCUSSION - Why Look at Old Problems?

Problems solved in the past can provide information that is still important and useful today. The entire past problem and its solution may not be exactly the same as the current problem. However, when aspects of the past problem are similar to aspects of the current problem, aspects of the past problem's solution can support the current problem's solution.

The slide shows the concept that by using knowledge gained from analyzing older aircraft designs, engineers in the present can design new aircraft. The old aircraft were designed to support problems entirely different from the new aircraft design, but certain aspects of the new design (e.g., carry people, carry cargo, need for engines) are the same as those of the old design.

STUDENT INTERACTIONS

- Can you give some examples of products that were designed in the past that are still being made now? Are they better now?
- Name some situations where you had to look at some things you did in the past to help you do something now.

OBJECTIVE

Students should be able to understand that problems solved in the past have solutions that can be reused today and even in the future.

3-2

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Why Look at Old Problems?

- Old problems have solutions that new problems can use.

Old Aircraft

Design Team

New Aircraft

SOFTWARE
PRODUCTIVITY
CONSORTIUM
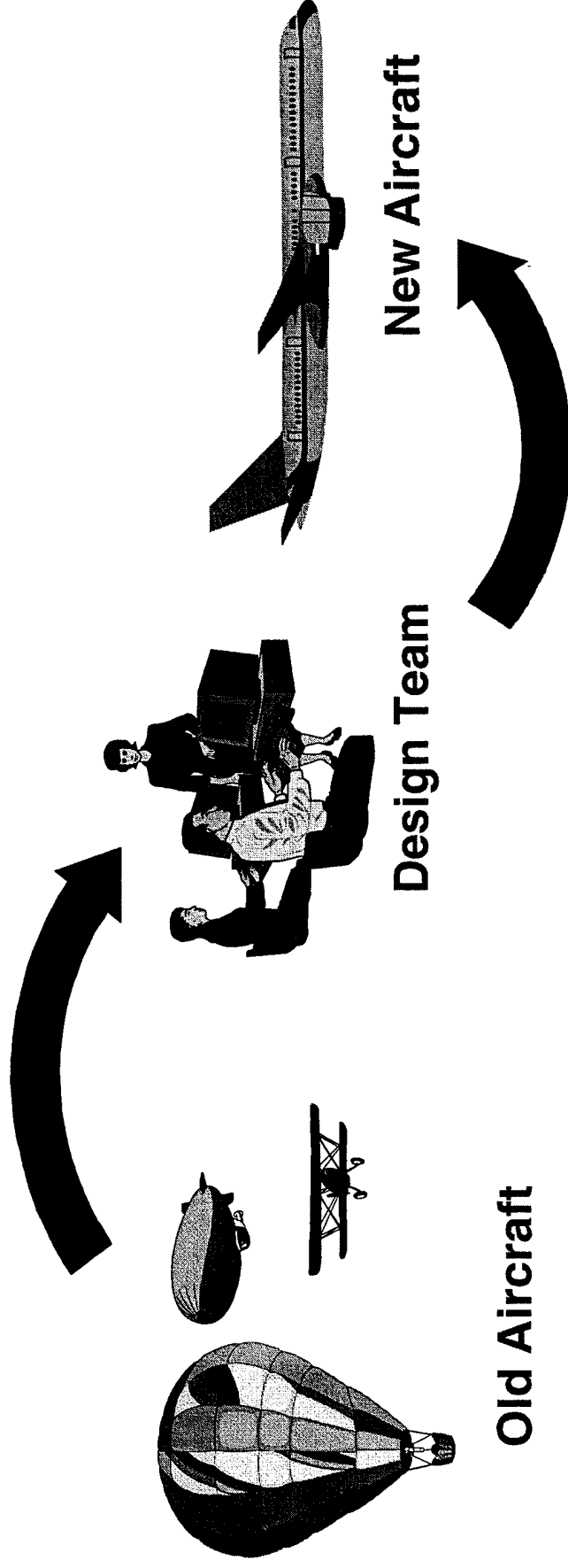
DISCUSSION - Look at Solutions

If you look at the old problems (the domain), they contain pieces (or systems) that can be analyzed. These systems are the key to identifying the reusable functions. You model these individual systems to be able later to identify both the common and different functions.

The slide shows the domain of telephone systems. This domain consists of a number of telephone designs, or systems. These systems must be modeled individually so the reusable functions can be defined.

STUDENT INTERACTION

If you have a domain called mathematics, what are the different tools, or systems, you can use to do mathematics?
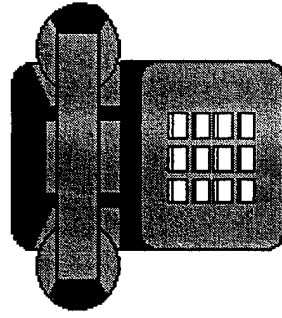
OBJECTIVE

Students should be able to analyze a domain and, at a high level, be able to decompose the domain into its supporting systems.
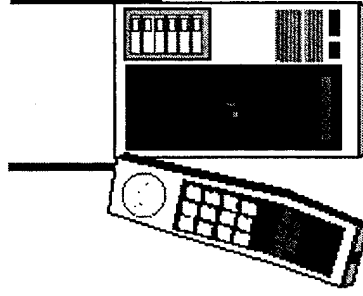
Look at Solutions

- Examples of systems that make up a domain

Domain: Telephone Systems

Cellular Telephone

Cordless Telephone

Corded Telephone

3-3

SOFTWARE
PRODUCTIVITY
CONSORTIUM

DISCUSSION - Decompose Solutions

In domain analysis, you decompose the systems into their individual requirements and then into their supporting functions and unique qualities. In most cases, you will not be able to break down the requirements because they probably are not documented. You can begin, however, to decompose the functions. As you decompose the information in each system, you identify the details of the system (i.e., functions, features, interface requirements).

If you are dealing with physical systems, you decompose them into their physical parts. Each part fulfills a high-level function.

In the example, we decompose a corded phone into its supporting physical parts. The phone is composed of a handset, cord, keypad, and a base (or housing).

STUDENT INTERACTIONS

- Can you decompose the calculator into its individual functions? What are the different things a calculator can do?
- Can you decompose the corded telephone into its individual functions?
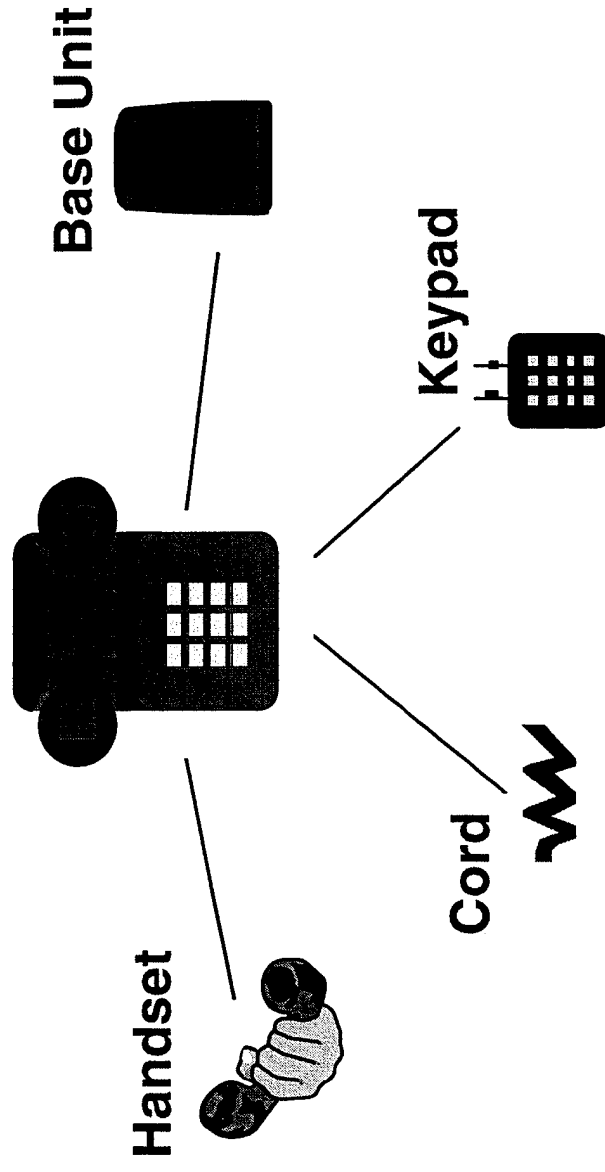
OBJECTIVE

Students should be able to explain the meaning of decomposing a system into its component functions.

Decompose Solutions

- First identify the system.
- Next identify the pieces that are components of the system.

System: Corded Telephone

Handset

Cord

Keypad

Base Unit

DISCUSSION - Important Rules

Since the models you create are for individual systems that already exist, you should model only the information you have. This means that if you model a function in system 1 and you do not see it in system 2, do not impose that function on system 2.

Also, do not add new functions that you or the user desire. This can be incorporated later in the domain analysis process. At this time, the emphasis is on **existing functions and components only**.

STUDENT INTERACTIONS

- Why should you not add new functions at this time?
- Discuss the two rules for this step; do not impose and do not add.

OBJECTIVE

Students should be able to understand that they must limit the amount of information included in each model to the system's current capabilities.

3-5

Important Rules

- Model existing system and its components.

- Do not introduce any new requirements or components.

Correct Model

System: Corded Telephone

Base Unit

Keypad

Handset

Cord

Incorrect Model

System: Corded Telephone

Base Unit

Keypad

Handset
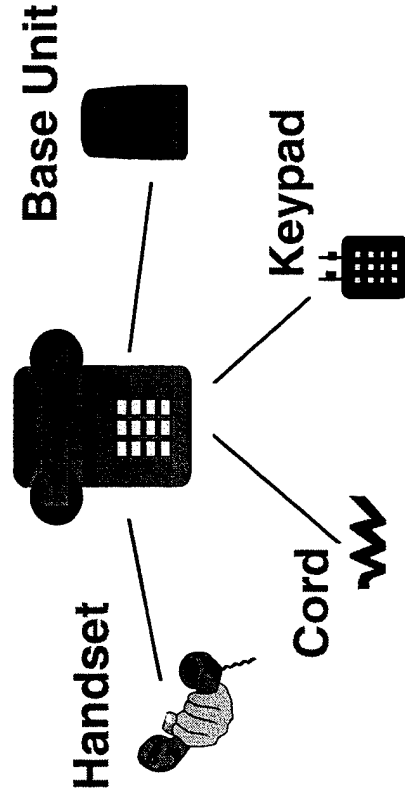
Cord

DISCUSSION - Gather Information

To analyze the various systems that represent old problems, the engineer must gather various types of information on both the domain and the systems within the domain. Information sources include:

- Concept of operations document for a system
- Requirements specifications sheets for the system
- Design documents
- Existing code
- Interviews with the software developers who designed the old system(s)
- Interviews with the users of the old system(s)

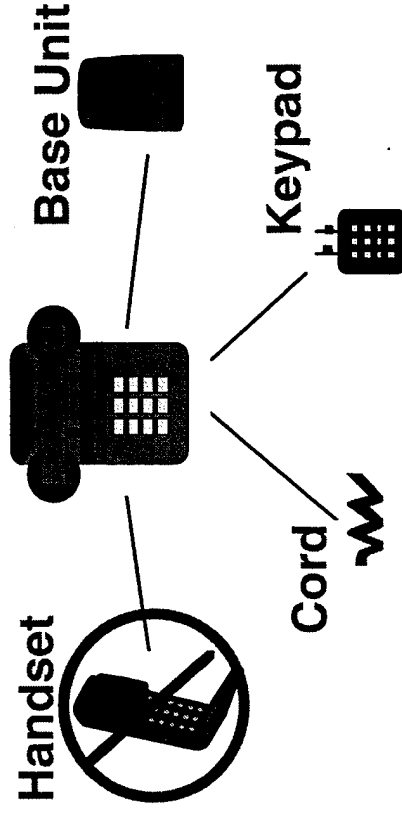The slide shows the various sources of information the engineer has at his or her disposal. Practice has shown that the two most useful sources of information are the software developers who designed the system and the users of the system.

STUDENT INTERACTIONS

- If you were about to build a model airplane or rocket, what type of information sources would you seek?
- If you were required to write a report on the international space station, what kinds of information would you look for? What type(s) of information sources would you seek?
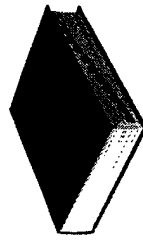
OBJECTIVE

Students should be able to understand the importance of finding and storing information to solve a problem.
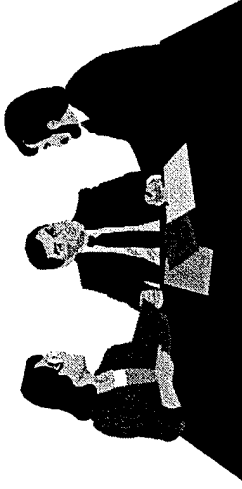
3-6

Gather Information

- Research and identify available sources of information on the systems within the DOF.

- Store the information.

Software

Interviews

Internet

Books/Documents

3-6
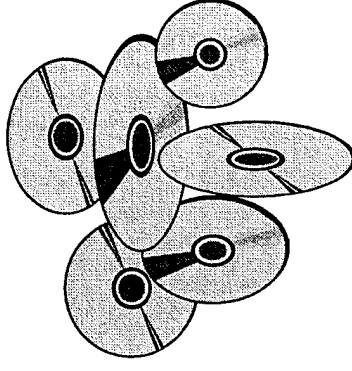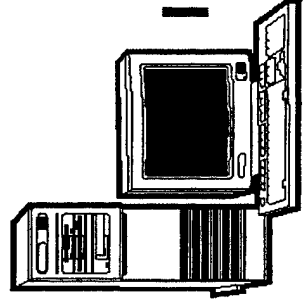
DISCUSSION - Too Many Systems!!

The key to completing a successful analysis is choosing a set of systems that represent the entire gamut of the capabilities of systems within the domain. In domain analysis, this collection of select systems is known as the representative set. Ideally, if you choose all systems within the domain to analyze, you will have the best results for the effort. However, the reality is that the analysis team has only so many resources (people, money, and equipment) and must work within a prescribed schedule. Even under these constraints, the domain analyst must choose a set of systems that properly represent the domain.

The selection of the representative set is a very difficult process. Usually domain experts assist the domain analysts in arriving at the representative set; sometimes the representative set must be estimated by the analyst.

STUDENT INTERACTION

If you had to write a report for the mathematics domain we discussed earlier, and you were asked to pick only three math tools to use, what would they be? Why select those rather than others?

OBJECTIVE

Students should be able to understand that a number of ways may exist to solve a problem. Because of real-world limitations, not all of these potential solutions may be appropriate.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Too Many Systems!!

- **Be smart: Choose only what you need.**

Cordless Telephone
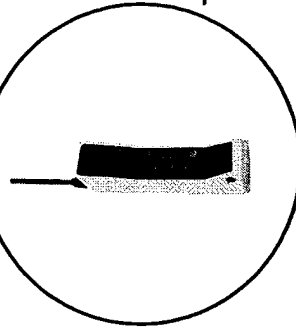
Corded Telephone

900 MHz Cordless Telephone
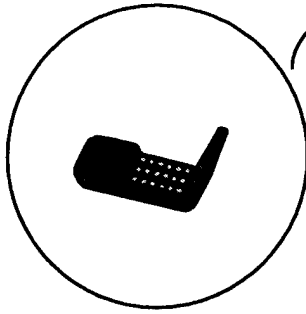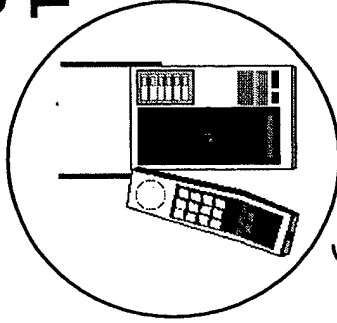
Cellular Telephone
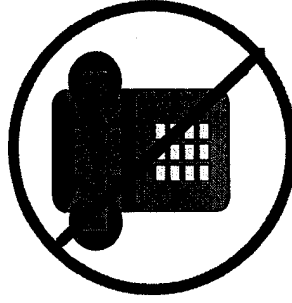
My problem is to design a phone that a user can carry a given distance. I think these three systems will give me enough information.

3-7

DISCUSSION - Model the Information for Each System

Once you have identified the domain, the systems within the domain, and the domain information sources, all that is left is to model the information. To do this, domain analysts must ask many questions, some of which are:

- What functions does the system perform?
- How do the functions relate back to requirements?
- How does the function work with other functions and users?
- What makes the function work?
- What makes the function not work?
- What information does it need and send?

As this information is generated, domain analysts create functional models to represent it. The key is to model the information using categories broad enough to represent many types of items. As the slide shows, the model usually is created with the domain on top; below that are listed requirements and the decomposition of those requirements, and further decomposition of those requirements into their functions and subfunctions. Of course, the depth of information varies from requirement to requirement and from function to function. This aspect of the modeling is not a science, but an art. Good judgment and creative thinking are required.

STUDENT INTERACTION

Go through simple examples in which the students can decompose some systems (e.g., TV sets, stereo).

OBJECTIVE

Students should be able to model various types of information into logical categories.

Model the Information for Each System

- Summarize and organize the information into functional models.

A System in the Domain

Requirement 1
- Function 1
- Function 2

Requirement 2
- Subrequirement 1
  - Function 1
    - Subfunction 1.1
    - Subfunction 1.2
  - Function 2
- Subrequirement 2

3-8
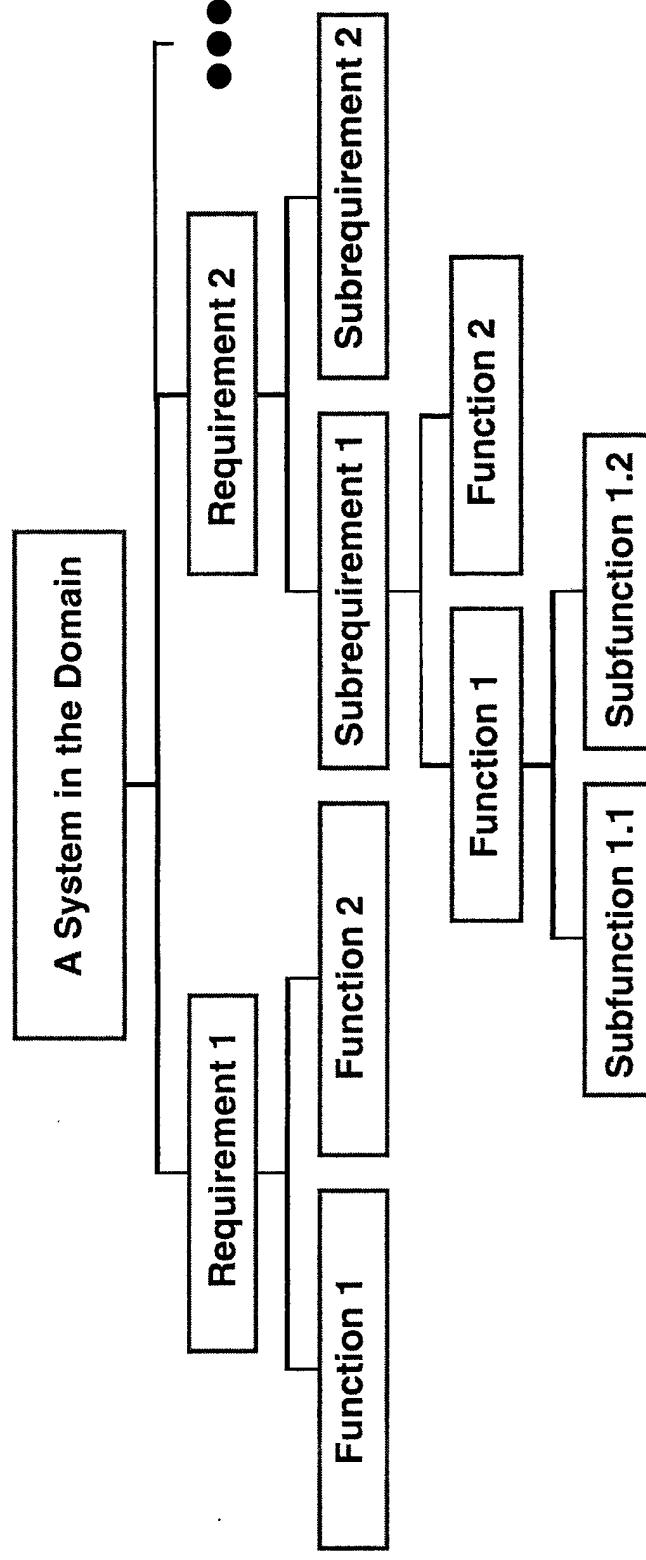
DISCUSSION - Example: Remote 1

The next two slides depict two individual systems for the domain of remote controller. There are two types of remote controllers: one for a TV and the other for a stereo system, which we call Remote 1 and Remote 2, respectively. We will model the functions for each remote individually. Consider these remotes to be old solutions to a similar problem; the user requires a method to control both the TV and stereo without actual contact with the units.

For the sake of simplicity, we will skip modeling the requirements and go directly to modeling the functions. As the example shows, the TV remote has certain basic functions: control the volume, turn power on and off, and control the channel. It also has certain advanced functions: control the VCR, choose to display closed captions or not, and view all the display functions (color, tint, brightness, etc.) from the remote.

SEE NEXT PAGE FOR STUDENT INTERACTION AND OBJECTIVE

3-9

Example: Remote 1

- A TV system consists of the following functions:

TV Remote Control

Select Basic Features

Select Advanced Features

Control Volume

Control Channel

Power

Display Closed Captions

Control Display

Control VCR

SOFTWARE PRODUCTIVITY CONSORTIUM
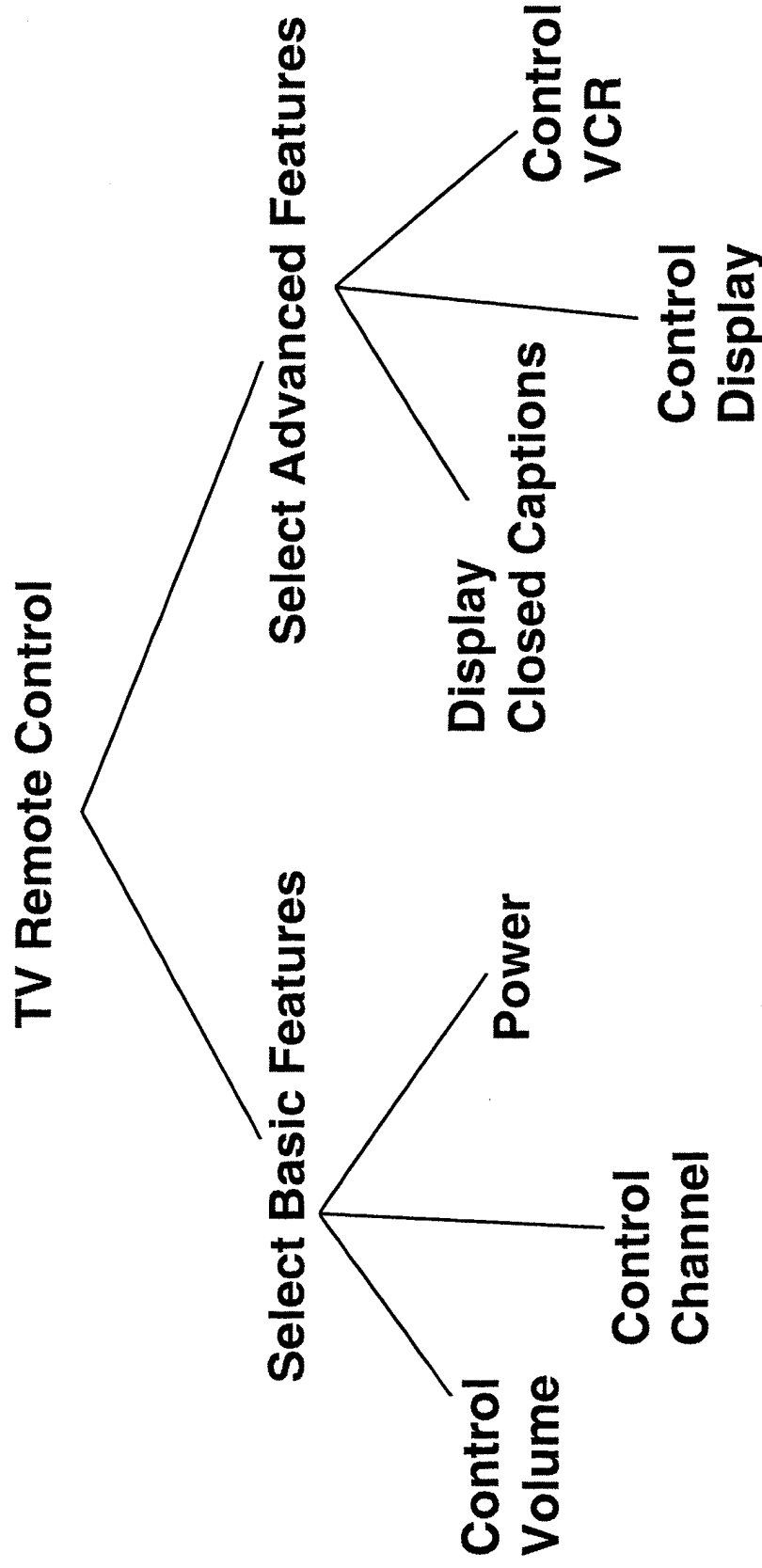
DISCUSSION -Example: Remote 2

The stereo remote also has certain basic functions:

- Control the volume
- Turn power on and off
- Select the type of media to play (CD, tape, laserdisc, tuner)
- Control station selection

Furthermore, the stereo remote has advanced features:

- Turn surround sound on and off
- Control the CD, tape, laserdisc, and tuner from the same remote

STUDENT INTERACTION

Look at the decomposition of the two remote controllers. Can you decompose these functions further?

OBJECTIVE

Students should be able to model the features and functions of various systems.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Example: Remote 2

- The stereo system remote has the following functions:

Stereo Remote Control

Select Basic Features

Select Advanced Features

Use Power

Select Media Type

Control Station

Control Volume

Select Surround Sound

Control Display

Provide Media Functions

3-10
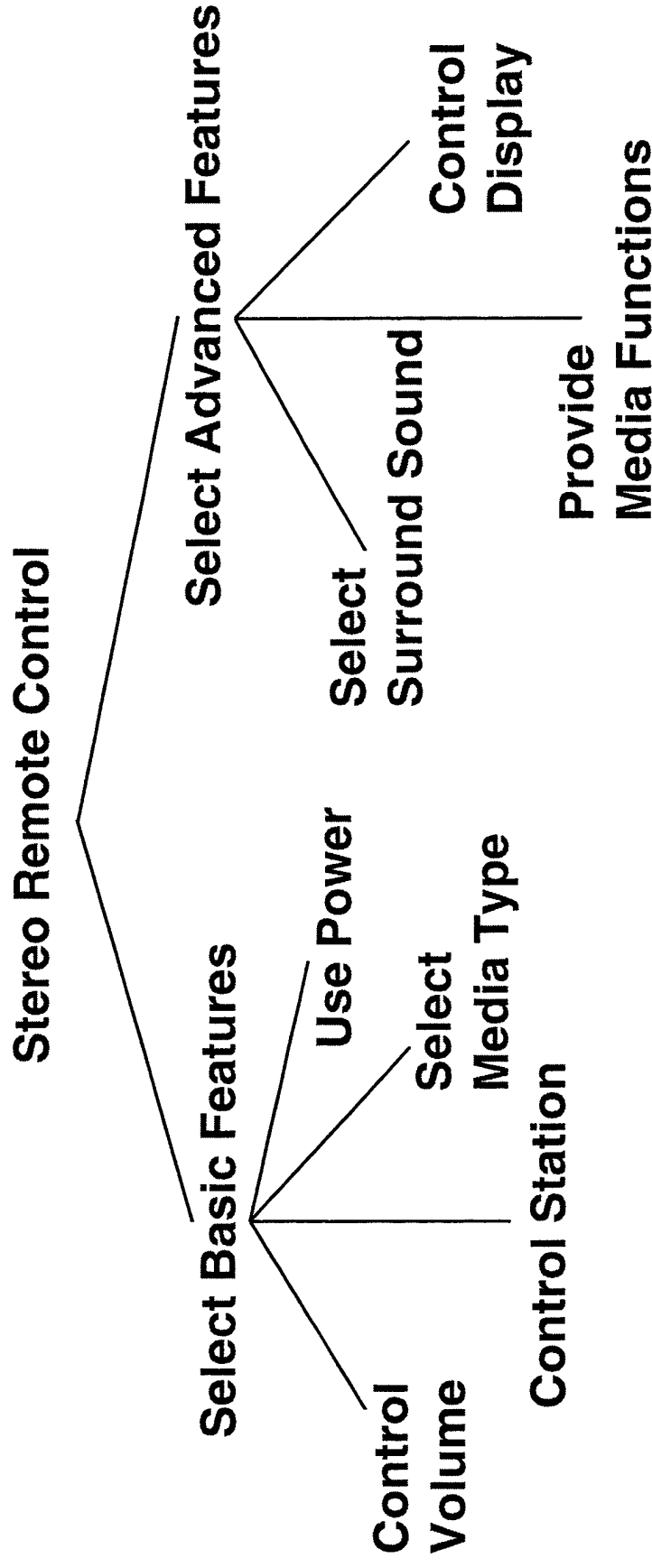
DISCUSSION - Unit Summary

Problems solved in the past can provide information that is still important and useful today. The entire past problem and its solution may not be exactly the same as the current problem. However, when aspects of the past problem are similar to aspects of the current problem, aspects of the past problem's solution can support the current problem's solution.

An individual system model describes the individual system of a past problem and represents it as a graphical model. The key is to model as many systems as necessary to describe effectively the common and different requirements and functions within a domain.

OBJECTIVES

Students should be able to:

- Explain why knowledge about past problems is important
- How to develop an individual system model

3-11

Unit Summary

- Individual system models:

  — Represent the common and different functions and requirements of systems within a domain

  — Represent the current system, not new systems

SOFTWARE
PRODUCTIVITY
CONSORTIUM

UNIT 3: INDIVIDUAL SYSTEM MODELS

Summary

This unit describes the second major activity of domain analysis, domain modeling. In domain analysis, the detailed analysis—finding both the common and different aspects of the domain—is accomplished.

Problems solved in the past can provide information that is still important and useful today. The entire past problem and its solution may not be exactly the same as the current problem. However, when aspects of the past problem are similar to aspects of the current problem, aspects of the past problem's solution can support the current problem's solution.

If you look at the old problems (the domain), they contain pieces (systems) that can be analyzed. These systems are the key to identifying the reusable functions. You model these individual systems to be able later to identify both the common and different functions.

In domain analysis, you decompose the systems into their individual requirements and then into their supporting functions and unique qualities. In most cases, you will not be able to break down the requirements because they probably are not documented. You can begin, however, to decompose the functions. As you decompose the information in each system, you identify the details of the system (e.g., functions, features, interface requirements).

Since the models you are creating are for individual systems that already exist, you should model only the information you have. This means that if you model a function in system 1 and you do not see it in system 2, do not impose that function on system 2. Also, do not add new functions that you or the user desire. This can be incorporated later in the domain analysis process. At this time, the emphasis is on **existing functions only.**

To analyze the various systems that represent old problems, the engineer must gather various types of information on both the domain and the systems within the domain. Many information sources exist. Practice has shown that the two most useful sources of information are the software developers who designed the system and the users of the system.

The key to completing a successful analysis is choosing a set of systems that represent the entire gamut of the capabilities of systems within the domain. In domain analysis, this collection of select systems is known as the representative set. Ideally, if you choose all systems within the domain to analyze, you will have the best results for the effort. However, the reality is that the analysis team has only so many resources (people, money, and equipment) and must work within a prescribed schedule. Even under these constraints, the domain analyst must choose a set of systems that properly represent the domain.

The selection of the representative set is a very difficult process. Usually domain experts assist the domain analysts in arriving at the representative set; sometimes the representative set must be estimated by the analyst.

Once you have identified the domain, the systems within the domain, and the domain information sources, all that is left is to model the information. To do this, domain analysts must ask many questions and use the answers as model information.

As this information is generated, domain analysts create functional models to represent it. The key is to model the information using categories broad enough to represent many types of items.

The model usually is created with the domain at the top; below that are listed requirements and the decomposition of those requirements, and further decomposition of those requirements into their functions and subfunctions. Of course, the depth of information varies from requirement to requirement and from function to function. This aspect of the modeling is not a science, but an art. Good judgment and creative thinking are required.

An individual system model describes the individual system of a past problem and represents it as a graphical model. The key is to model as many systems as necessary to describe effectively the common and different requirements and functions within a domain.

Exercises

Exercise 1

Pretend that you belong to a hobby company that manufactures radio controllers for the following radio-controlled toys:

- Cars

- Aircraft

- Ships

List some common and different functions for a function called Steering in the domain of transportation. List several different ways that steering is performed in the three domains.

Exercise 2

In the area of audio components are the following domains:

- Portable tape players

- CD players

- AM/FM stereos

Ask each student to select one of these devices. They are to go home and look at the device and read the manual (if it is still available) to determine the functions the device performs. Then they are to list the functions each device performs and model the information into a functional model as described In the class lectures.

Exercise 3

This is a continuation of Exercise 3, Unit 2.

Using the context diagram as a beginning model, begin to create the individual DOF system models for each robot (based on the DOF and the requirements specifications sheets).

The individual system models will look very similar to the model created in Unit 2. For each robot the top of the tree will be the DOF. In this case, it is function Move_Robot_Body_Parts. Now, as you review more information about the various body parts for each robot, you will need to identify the components, or code, that compose the Move_Robot_Body_Parts. These are the actual code modules that allow each robot to manipulate various joints on its body.

Teacher's Notes for Exercises

The following are notes for the teacher to apply to the exercises presented in this unit. The lists need not be comprehensive, but should be adequate to develop the students' understanding of the material.

Exercise 1

Pretend that you belong to a hobby company that manufactures radio controllers for the following radio-controlled toys:

- *Cars*

- *Aircraft*

- *Ships*

List some common and different functions for a function called Steering in the domain of transportation. List several different ways that steering is performed in the three domains.

For cars, the function of Steering can be decomposed into the following: steer right and steer left in forward and reverse.

Car Steering Model

Steer_Car
Steer_Right
Steer_Left
Go_Forward
Go_Reverse
Go_Forward
Go_Reverse

For aircraft, the function of Steering can be decomposed into the following: In the air, you can steer right, steer left, pull up, go down, bank right, and bank left. On the ground, the functions are the same as for cars.

Aircraft Steering Model

Steer_Aircraft

Steer_Air

Steer_Ground

Steer_Right

Bank_Right

Steer_Right   Steer_Left

Steer_Left

Pull_Up

Bank_Left

Go_Down

For ships, the function of Steering can be decomposed into the same functions as for cars.

Ship Steering Model

Steer_Ship

Steer_Right

Steer_Left

Go_Forward   Go_Reverse   Go_Forward   Go_Reverse

Exercise 2

In the area of audio components are the following domains:

- *Portable tape players*

- *CD players*

- *AM/FM stereos*

Ask each student to select one of these devices. They are to go home and look at the device and read the manual (if it is still available) to determine the functions the device performs. Then they are to list the functions each device performs and model the information into a functional model as described In the class lectures.

For portable tape players, the following functions can apply: play tape, rewind tape, forward tape, pause tape, stop tape, search music forward, search music reverse, record on tape, erase tape, increase tape speed, decrease tape speed, increase volume, decrease volume, mute volume, select music sound.

Functional Model: Portable Tape Player

Use_Portable_Tape_Player

Find_Music

Adjust_Volume

Search_Music_Reverse

Search_Music_Forward

Increase_Volume   Mute_Volume

Decrease_Volume

Use_Other_Functions

Decrease_Tape_Speed   Erase_Tape

Increase_Tape_Speed

Select_Music_Sound

Listen_To_Tape

Record_On_Tape

Play_Tape

Stop_Tape   Pause_Tape

Rewind_Tape   Forward_Tape

For CD players, the following functions can apply: play CD, skip CD forward, skip CD reverse, pause CD, stop CD, search music forward, search music reverse, increase volume, decrease volume, mute volume, select music sound.

Functional Model: CD Player

Use_CD_Player

Find_Music

Adjust_Volume

Increase_Volume   Mute_Volume

Decrease_Volume

Search_Music_Forward

Search_Music_Reverse

Listen_To_CD

Use_Other_Functions
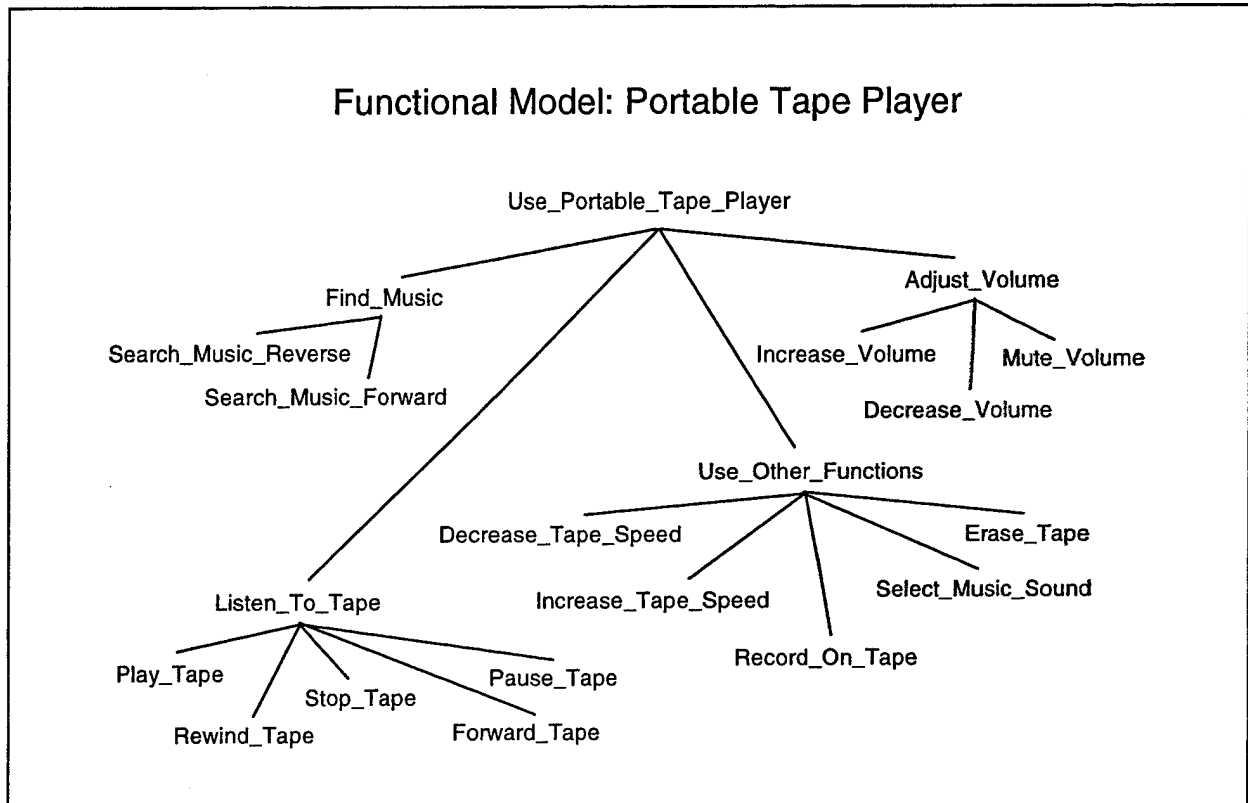
Play_CD   Pause_CD

Stop_CD

Skip_CD_Forward   Skip_CD_Reverse

Select_Music_Sound
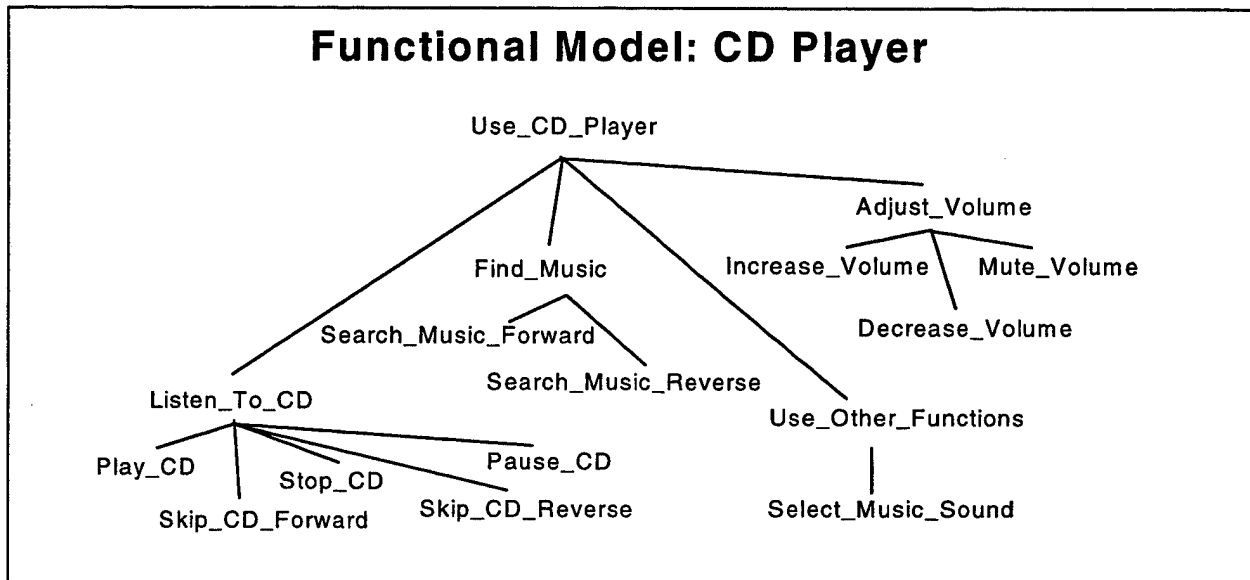
For AM/FM stereos, the following functions can apply: tune station forward, tune station backward, scan stations, seek stations, save station to memory, increase volume, decrease volume.

Functional Model: AM/FM Stereo Tuner

Use_Tuner

Listen_Tuner

Tune_Station_Forward

Select_AM/FM

Tune_Station_Reverse

Find_Music

Scan_Stations   Seek_Stations

Adjust_Volume

Increase_Volume   Mute_Volume

Decrease_Volume

Use_Other_Functions

Save_Station_Memory

Exercise 3

This is a continuation of Exercise 3 from Unit 2.

Ask the students to review the requirements specifications sheets and derive the following functional models based on the requirements:

Model 1

Individual System Model: Agricultural Robot

Move_Ag_Robot_Body_Parts

- Move_Entire_Body
  - Rotate_Body
    - Rotate_Right
    - Rotate_Left
  - Roll_Body_4Wheels
    - Forward_Speed
    - Reverse_Speed
  - Extend_Legs_Independently
    - Ascend_Body
    - Decend_Body
- Move_Fingers
  - Open_Fingers
  - Close_Fingers
- Move_Arms
  - Rotate_Single_Arm_Up
  - Rotate_Single_Arm_Down

Model 2

Individual System Model: Search and Rescue Robot

Move_S_and_R_Robot_Body_Parts

- Move_Entire_Body
  - Rotate_Body
    - Rotate_Right
    - Rotate_Left
  - Roll_Body_Tracks
    - Roll_Track_Right
    - Roll_Track_Left
  - Extend_Legs_Independently
    - Ascend_Body
    - Decend_Body
- Move_Fingers
  - Open_Fingers
  - Close_Fingers
- Move_Arms
  - Rotate_Multi_Arm_Up
  - Rotate_Multi_Arm_Down
  - Extend_Arms_Away
  - Extend_Arms_Toward

Model 3

Individual System Model: SanitationRobot

Move_San_Robot_Body_Parts

Move_Entire_Body

Move_Arms

Rotate_Body

Rotate_Multi_Arm_Up

Rotate_Right   Rotate_Left

Move_Fingers

Rotate_Multi_Arm_Down

Roll_Body_Tracks

Open_Fingers   Close_Fingers

Extend_Arms_Away

Roll_Track_Right   Roll_Track_Left

Extend_Arms_Toward

DISCUSSION - Unit 4: Domain Modeling, Step 2: Descriptive Models

In Unit 3 we discussed the modeling of past problems and their unique solutions. These solutions provide the engineer with insight into how and why the systems within a domain were designed.

This unit shows you how to examine the individual system models and identify the common and different functions. You will create a new model by combining all the common and different functions.

OBJECTIVES FOR THE UNIT

Students should be able to:

- Define the terms common functions and different functions
- Meld individual system models into one model that includes both the common and different functions (also known as a descriptive model)

Unit 4: Domain Modeling
Step 2: Descriptive Model

SOFTWARE
PRODUCTIVITY
CONSORTIUM

DISCUSSION - What Is a Descriptive Model?

A combined, or descriptive, model is a graphical representation of the common and different functions, features, and requirements of each individual system model. This is the key model that domain design uses to create generic reusable software components. During domain design, the common functions are organized further into software modules that represent routines that a software system must perform. The goal of domain analysis is to create descriptive models for all the systems in a domain so all software functions can be identified.

In the example we look at a domain called computing. In this domain, we have identified two types of computer systems available to users: desktop systems and laptop systems. Let's model these as individual systems (see previous unit for a discussion of individual system models). At a high level, their parts are the same, based on certain requirements. Common parts are the keyboard, display, base unit (CPU), and pointing device. From a high level, there seem to be no real differences.

STUDENT INTERACTIONS

- Can you look at the computing domain example and explain where the differences lie?
- We have discussed some domains in class. Which functions would you combine to create a descriptive model? Can you explain why you would combine them in that way?

OBJECTIVE

Students should be able to understand why one would want to combine the various individual models.

4-2

What Is a Descriptive Model?

- Graphic of all common and different functions identified from individual system models

Laptop System

Display

Pointing Device

Keyboard

Base Unit

Desktop System

Pointing Device

Keyboard

Display

Base Unit
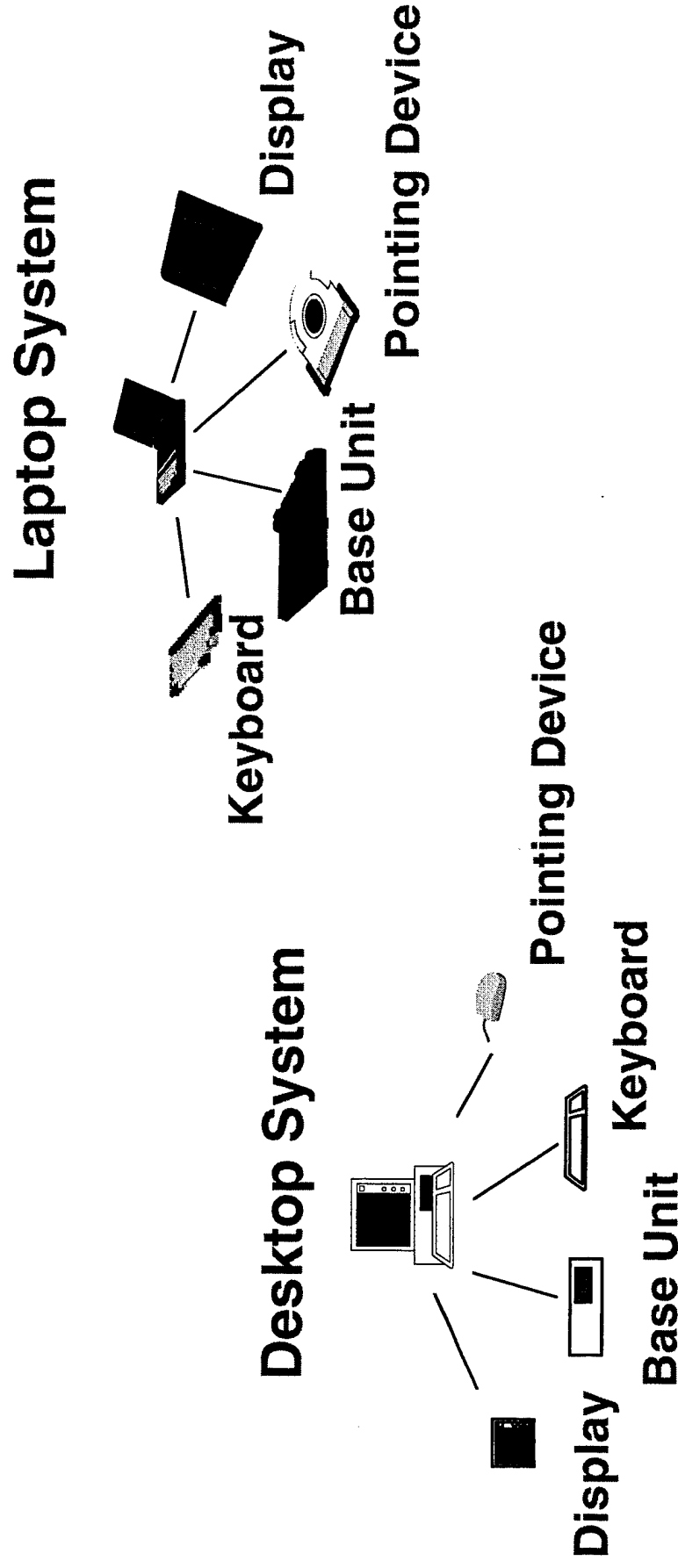
DISCUSSION - Why Create a Descriptive Model?

The point of creating a descriptive model is to represent the knowledge of past problems that have been solved. This is accomplished by gathering all requirements, supporting functions, features, and other aspects of the various systems in the domain. As the information is derived, it must be carefully analyzed and categorized into common and different items.

When the requirements or functions of a new system match the requirements or functions of the systems in the descriptive model, this information is then used to design new software solutions. A key aspect of creating this model is to allow traceability from the functions within the model back to the requirements that created them (as we discussed in the unit on individual system models). Using this information, the engineer can, with some effort, match the new system's requirements to the requirements of the older systems and identify the reusable functions, features, etc.

The slide shows the descriptive model for the computing domain created by combining the common and different parts of the desktop and laptop computers.

STUDENT INTERACTION

Ask the students to identify some ordinary household items and to categorize them into groups of common and different items.

OBJECTIVES

Students should be able to:

- Explain why descriptive models are necessary
- Explain the importance of traceability

4-3

Why Create a Descriptive Model?

- Arranges the possible common and different functions for existing systems

- Can help identify functions needed for a new system

Computing Domain

View Data   Calculate Data   Enter Data   Select Data Visually
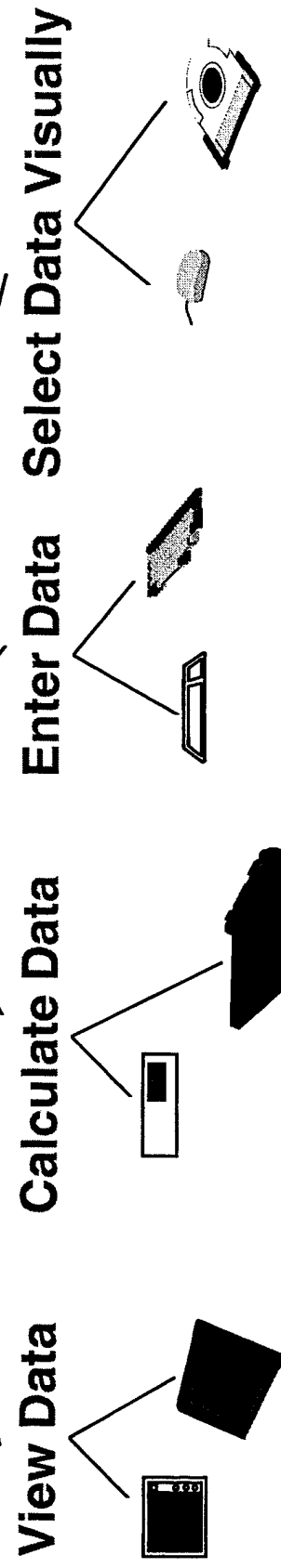
DISCUSSION - Warning!

So far we have discussed combining common and different items as if there were no concerns in doing so. There are, however, many concerns. At a high level, items could look as if they were common, but, at a lower level, the functions may actually be very different. Care must be taken not to categorize these functions into excessively broad generic groups.

As the slide shows, a fighter plane and a civilian aircraft, at a high level, have much in common. They both carry people; they both have wings; they both have critical avionics; and they both have engines. But if you decompose a part—for example, the wings—you will notice that a civilian aircraft has only a few wing configurations, while a fighter plane possesses many. Consequently, the designer would identify a high-level part called wings, but would possibly decompose it into minimum wing configurations and advanced wing configurations.

Software is much the same way. Lower-level details must be examined and grouped carefully.

STUDENT INTERACTION

Can you give some examples of functions for which the same term is used, but the term is used in different contexts?

OBJECTIVE

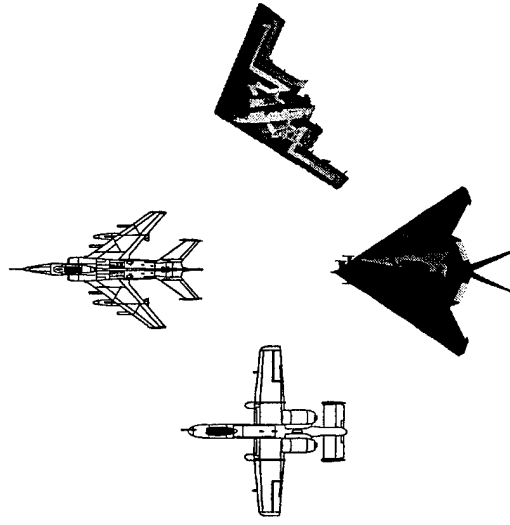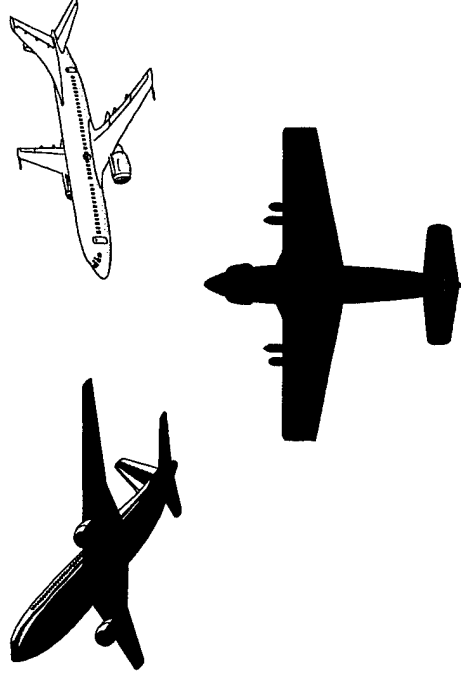Students should be able to explain why decomposing functions must be accomplished with great care.

4-4

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Warning!

- Sometimes common and different functions are not obvious.

Fighter Plane Wings

Civilian Plane Wings

4-4

DISCUSSION - Example: Remote Controller

In Unit 3 we created two individual system models, one for a TV remote and the other for a stereo remote. Now we will combine the individual models to create a common model of solutions to past problems. Again the key is to identify the common items so that the engineer can eventually arrange these common functions so they can be reused.

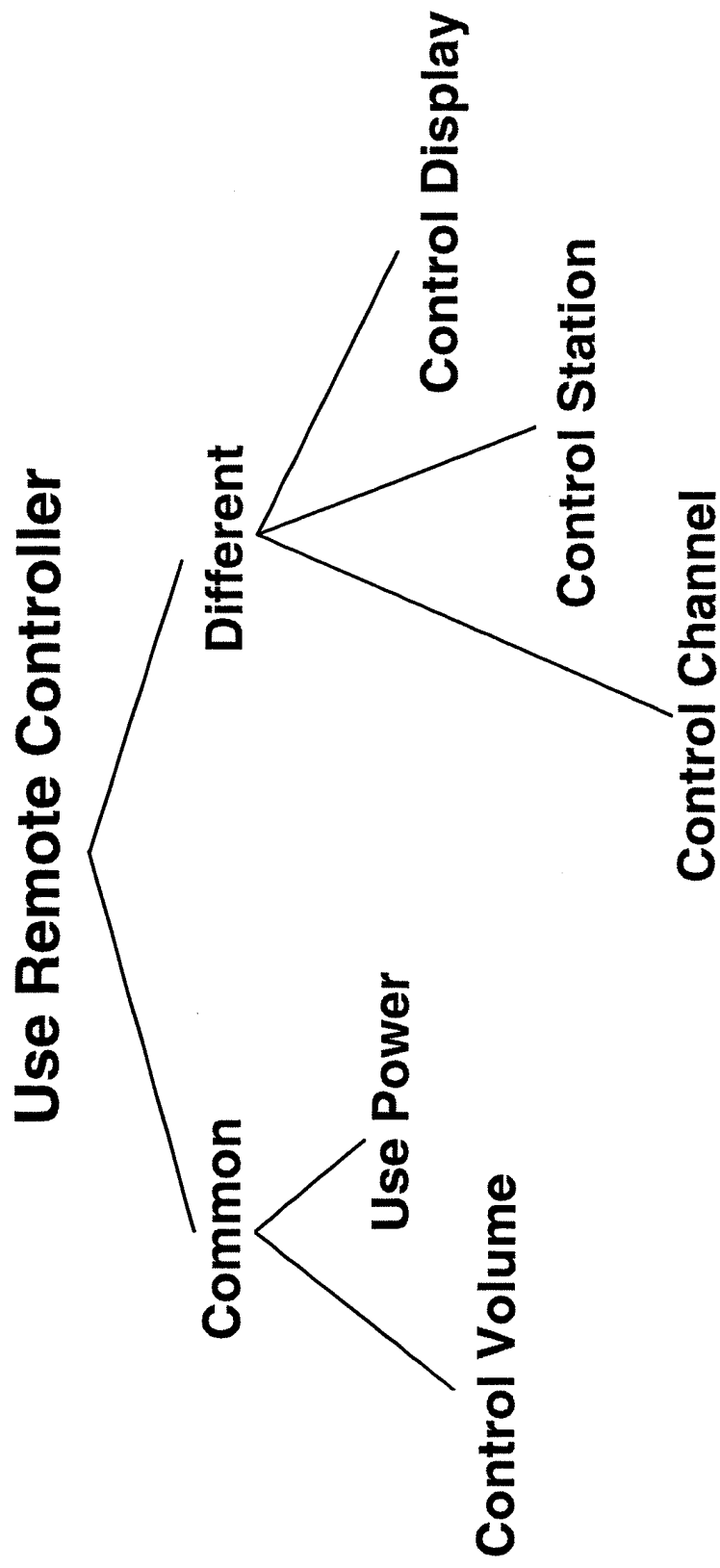STUDENT INTERACTIONS

- Prior to showing them this slide, ask the students to combine the two remote control models.
- Ask the students to decompose these models, demonstrating that this process is not a clear-cut matter of combining like terms.

OBJECTIVE

Students should be able to combine individual models to create a common model of solutions to past problems.

4-5

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Example: Remote Controller

Use Remote Controller

Common
- **Use Power**
- **Control Volume**

Different
- **Control Display**
- **Control Station**
- **Control Channel**

SOFTWARE PRODUCTIVITY CONSORTIUM

DISCUSSION - Unit Summary

A combined, or descriptive, model is a graphical representation of the common and different functions, features, and requirements of each individual system model. This is the key model that domain design uses to create generic reusable software components. During domain design, the common functions are organized further into software modules that represent routines that a software system must perform. The goal of domain analysts is to create descriptive models for all the systems in an entire domain so all software modules can be identified.

OBJECTIVES

Students should be able to:

- Describe the components of a descriptive model
- Identify how a descriptive model is used

4-6

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Unit Summary

- Descriptive model:

  — Combines common and different requirements and functions from individual system models.

  — Represents these elements in a single model.

  — Provides opportunity to identify components that can be reused in more than one system.

4-6

UNIT 4: DESCRIPTIVE MODELS

Summary

In this unit, we examine the individual system models and identify the common and different functions. A new model is created by combining all the common and different functions.

A combined, or descriptive, model is a graphical representation of the common and different functions, features, and requirements of each individual system model. This is the key model that domain design uses to create generic reusable software components. During domain design, the common functions are organized further into software modules that represent routines that a software system must perform. The goal of domain analysis is to create descriptive models for all the systems in a domain so all software functions can be identified.

The point of creating a descriptive model is to represent the knowledge of past problems that have been solved. This is accomplished by gathering all requirements, supporting functions, features, and other aspects of the various systems in the domain. As the information is derived, it must be carefully analyzed and categorized into common and different items.

When the requirements or functions of a new system match the requirements or functions of the systems in the descriptive model, this information is then used to design new software solutions. A key aspect of creating this model is to allow traceability from the functions within the model back to the requirements that created them (as presented in the previous unit in the disucussion of individual system models). Using this information, the engineer can, with some effort, match the new system's requirements to the requirements of the older systems and identify the reusable functions, features, etc.

So far we discussed combining common and variant terms as if there were no concerns in doing so. There are, however, many concerns. At a high level, items could look as if they were common, but, at a lower level, the functions may actually be very different. Care must be taken not to categorize these functions into excessively broad generic groups.

Exercises

Exercise 1

To continue the transportation exercise, list the common and different functions for the steering function for cars, aircraft, and ships. Group the functions by category. Finally, model the information functionally.

Exercise 2

To continue the audio components exercise, list the common and different functions for the portable tape player, CD player, and AM/FM stereo. Group the functions into categories. Finally, model the information functionally.

Exercise 3

Once individual models of the various types of robots are created, you will combine the models by melding the common items together. The result will be the common descriptive model. This model will identify the reusable pieces from the other robots that can be used to design a common requirements model and, later, the design specifications for the PEBOT.

To create the combined model, create a hierarchy of common functions within Move_Robot_Body_Parts. Identify the common modules and different modules from the individual models you created in the previous unit. A common module is a part that is shared by two or more robots. A different module is a part that is present in only one robot. Label the common and different modules in a way that identifies their characteristics.

Teacher's Notes for Exercises

The following are notes for the teacher to apply to the exercises presented in this unit. The lists need not be comprehensive, but adequate to develop the students' understanding of the material.
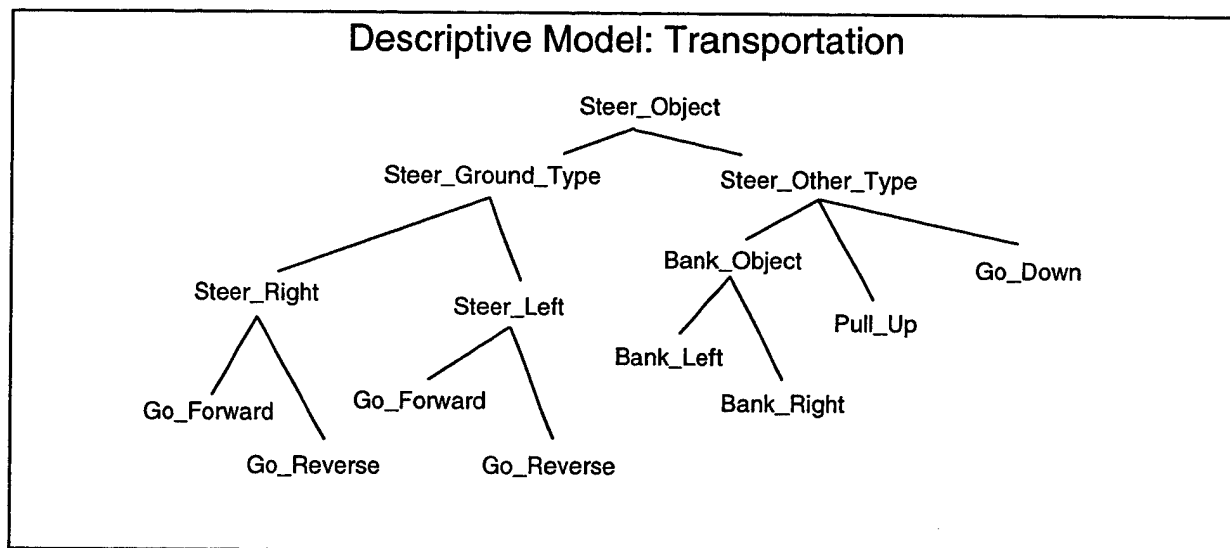
Exercise 1

To continue the transportation exercise, list the common and different functions for the steering function for cars, aircraft, and ships.

For automobiles, the function of steering can be decomposed into the following: steer right and steer left in forward and reverse.

For aircraft, the function of steering can be decomposed into the following: In the air, you can steer right, steer left, pull up, go down, bank right, and bank left. On the ground, the functions are the same as for cars.

For ships, the function of steering can be decomposed into the same functions as for cars.

By combining the common and different functions, students should arrive at the following descriptive model:
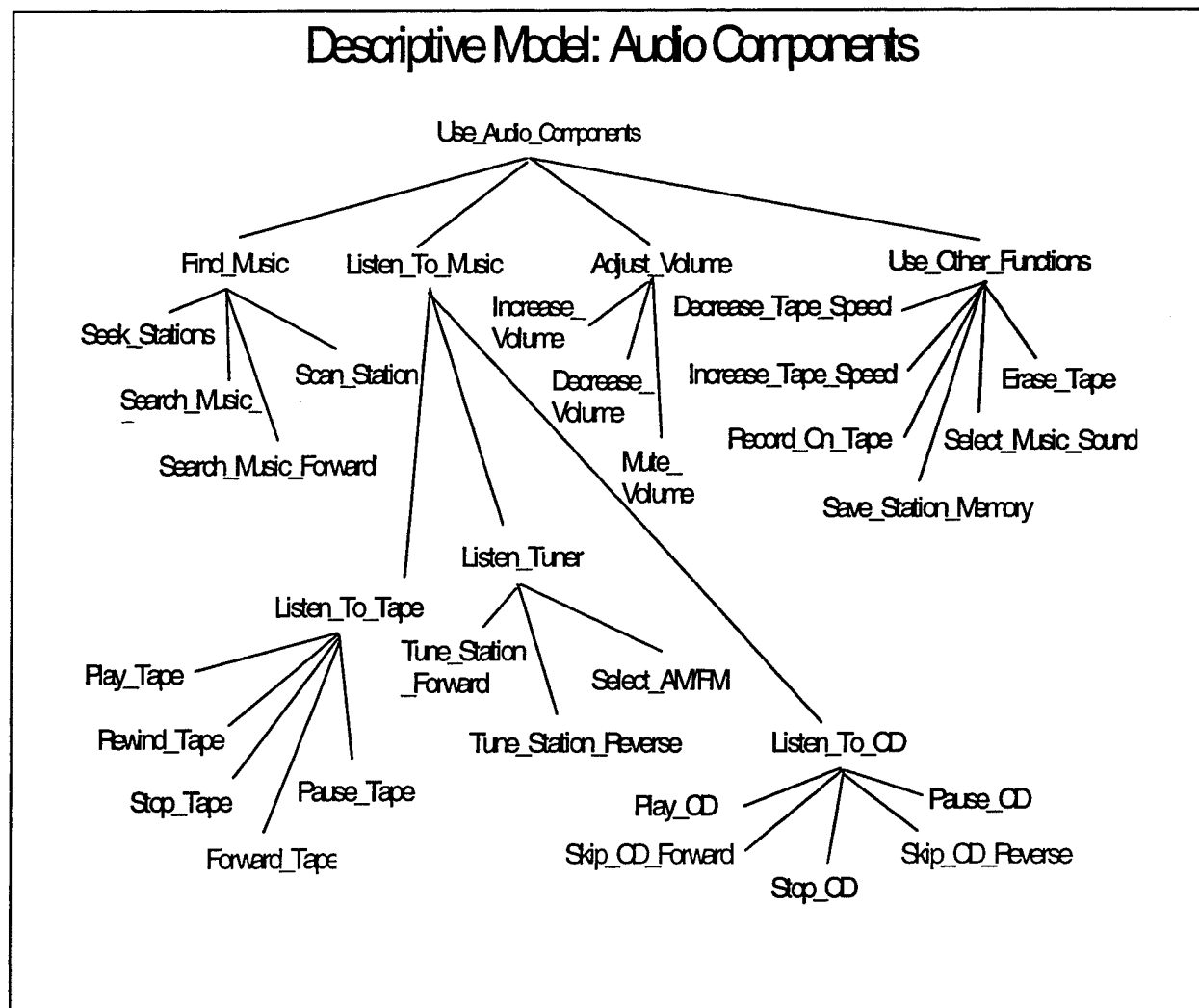

Descriptive Model: Transportation

Exercise 2

To continue the audio components exercise, list the common and different functions for the portable tape player, CD player, and AM/FM stereo. Group the functions by category. Finally, model this information functionally.

For portable tape players, the following functions can apply: play tape, rewind tape, forward tape, pause tape, stop tape, search music forward, search music reverse, record on tape, erase tape, increase tape speed, decrease tape speed, increase volume, decrease volume, mute volume, select music sound.

For CD players, the following functions can apply: play CD, skip CD forward, skip CD reverse, pause CD, stop CD, search music forward, search music reverse, increase volume, decrease volume, mute volume, select music sound.

For AM/FM stereos, the following can apply: tune station forward, tune station backward, scan stations, seek stations, save station to memory, increase volume, decrease volume.

By combining the common and different functions, students should arrive at the following descriptive model:

Descriptive Model: Audio Components

Use_Audio_Components

Find_Music   Listen_To_Music   Adjust_Volume   Use_Other_Functions

Seek_Stations
Search_Music
Scan_Station
Search_Music_Forward

Increase_Volume
Decrease_Volume
Mute_Volume

Decrease_Tape_Speed
Increase_Tape_Speed
Record_On_Tape
Erase_Tape
Select_Music_Sound
Save_Station_Memory

Listen_Tuner
Listen_To_Tape
Play_Tape
Rewind_Tape
Stop_Tape   Pause_Tape
Forward_Tape

Tune_Station_Forward
Select_AM/FM
Tune_Station_Reverse

Listen_To_CD
Play_CD   Pause_CD
Skip_CD_Forward   Skip_CD_Reverse
Stop_CD

Exercise 3

If students combine the individual system models for each robot in Exercise 3, Unit 3, they would create the following model:

Common Descriptive Model: Move_Robot_Body_Parts Function
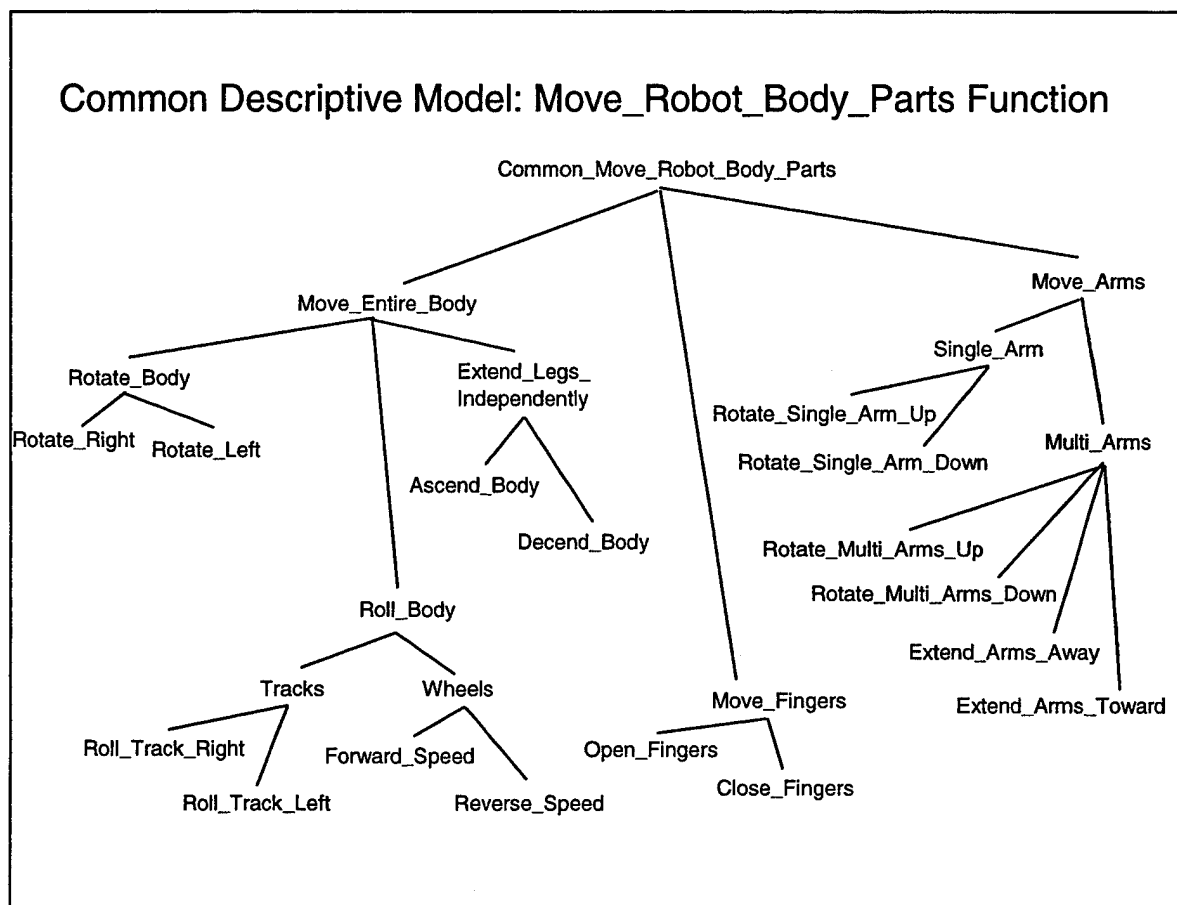
DISCUSSION - Unit 5: Domain Modeling, Step 3: Prescriptive Models

In Unit 4 we discussed the modeling of individual systems into a single model, known as a descriptive model. Each descriptive model is eventually handed off to domain design so that a generic solution can be designed for all systems in a domain.

The process described in this unit allows the domain analyst to enhance the descriptive model with features or functions that old solutions did not include. This model is referred to as the prescriptive model.

OBJECTIVE FOR THE UNIT

Students should be able to update or enhance the common or descriptive model to include features that a new system requires.

Unit 5: Domain Modeling Step 3: Prescriptive Models

SOFTWARE PRODUCTIVITY CONSORTIUM

DISCUSSION - What Is a Prescriptive Model?

The prescriptive model, also known as the to-be model, is simply an extension of the descriptive model. The descriptive model is created based on old problems and their unique solutions. When a new system is to be designed, certain functions are likely to overlap with the old functions; there is a need to create a descriptive model. The new system, however, will require updated functions, if not entirely new ones. The prescriptive model allows the engineer to estimate which updates or enhancements may be needed to accommodate future systems requirements.

Basically, a prescriptive model allows the domain analyst to model the best estimates for functions for new technology.

The slide shows how changing requirements have affected the design of data storage devices over a period of time and will continue to do so into the future. Initially data was stored on magnetic tapes housed in large machines. New requirements stated the need for a light and portable method to store data (the 5.25-inch diskettes). Later the requirement was to hold more information than 5.25-inch diskettes and to be smaller and less damage-prone (3.5-inch diskette). The updated requirement was to hold large amounts of data with quicker access (CD-ROM). Now the requirement is to hold even larger amounts of data and increase access time (optical disks). Similar changes will always be forthcoming.

STUDENT INTERACTIONS

- Can you think of how old TV sets have been updated or enhanced?
- Can you think of how the software products you use on your computer have been enhanced?

OBJECTIVE

Students should be able to understand what a prescriptive model is and what is its value added to the software engineering process.

5-2

SOFTWARE
PRODUCTIVITY
CONSORTIUM

What Is a Prescriptive Model?

- Model of the requirements for future system(s) in a domain
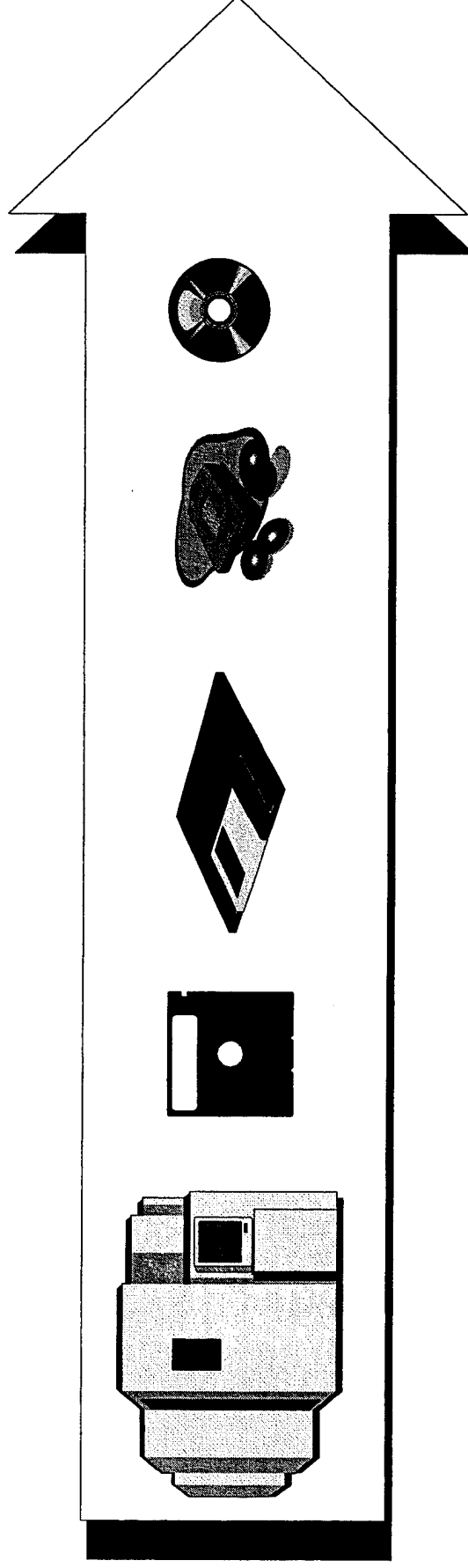
| Past Data Storage Requirement | Present Data Storage Requirement | Future Data Storage Requirement |

5-2

DISCUSSION - Why Use a Prescriptive Model?

The prescriptive model draws upon the domain analysts' and software developers' knowledge and creativity to enhance the existing descriptive model for future requirements and functions. The idea behind modeling old requirements and functions for use in designing a new system is to provide the software developers with a perspective into past systems that they may not possess. Once they gain this knowledge, they can design a new system with features from the old system(s), and can design it for reuse in future systems.

The slide shows that today most CDs are read access only, so a person can only access data, listen to it, etc. If you had a descriptive model of a CD system, you would have a requirement that said "read CD" or "listen to CD." If you were to design a new CD system and knew that a new requirement would be "write to CD," you would model that as an enhancing requirement. Therefore, on your prescriptive model, you would have a new branch called "write to CD."

STUDENT INTERACTIONS

- Can you think of how old VCRs have been updated or enhanced?
- Can you think of how the video games you use on your computer have been enhanced?
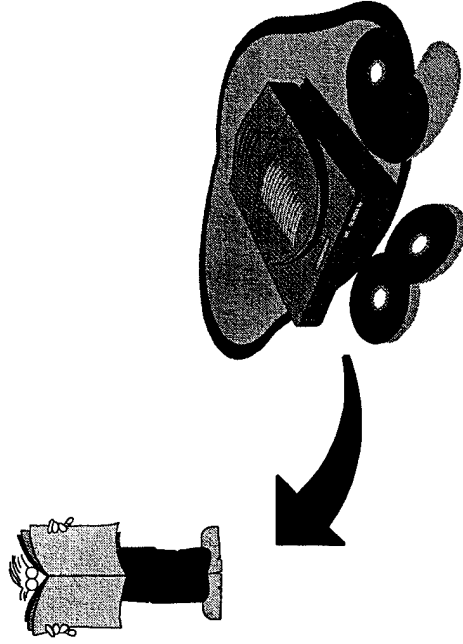
OBJECTIVE

Students should be able to understand the need for a prescriptive model.
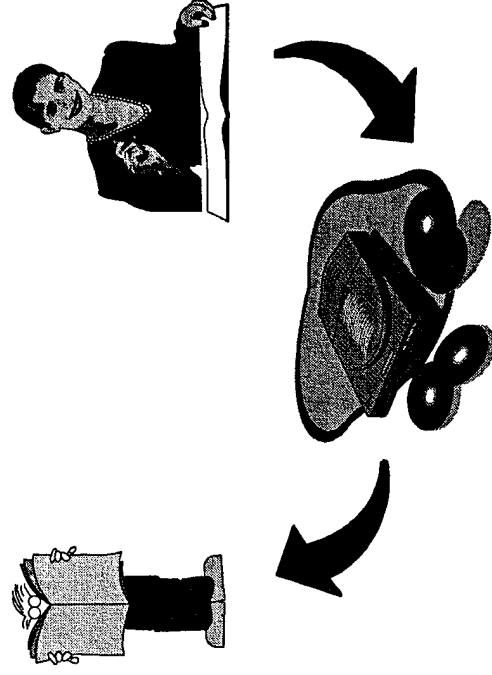
SOFTWARE
PRODUCTIVITY
CONSORTIUM

Why Use a Prescriptive Model?

- Present new functions that did not exist in older systems

**Current CDs are
Read Only**

**Future CDs need
to be Read and Write**

DISCUSSION - Using a Prescriptive Model

Once completed, the prescriptive model serves as the input to domain design. Domain design uses the model to create or package functions into bundles or modularized functions. These functions are sufficiently generic so the software developers can pick and choose the functions they need for a new system.

If you had many cars, trucks, vans, and buses broken down into pieces, and had grouped those pieces into common-element bundles, such as wheels, engines, doors, windows, seats, and transmissions, you theoretically could choose among the different pieces and design a new type of vehicle for a specific purpose. For example, you could take some pieces that make up a car, some pieces that make up a station wagon, and some pieces that make up a bus, and possibly create a minivan. Realistically, what you would do is to identify the requirements and functions of each, look for similarities, and then use this as a basis for your design. In the same way, software developers like to take various software systems, break them down into their component pieces, and select certain of the components to design a new software system.
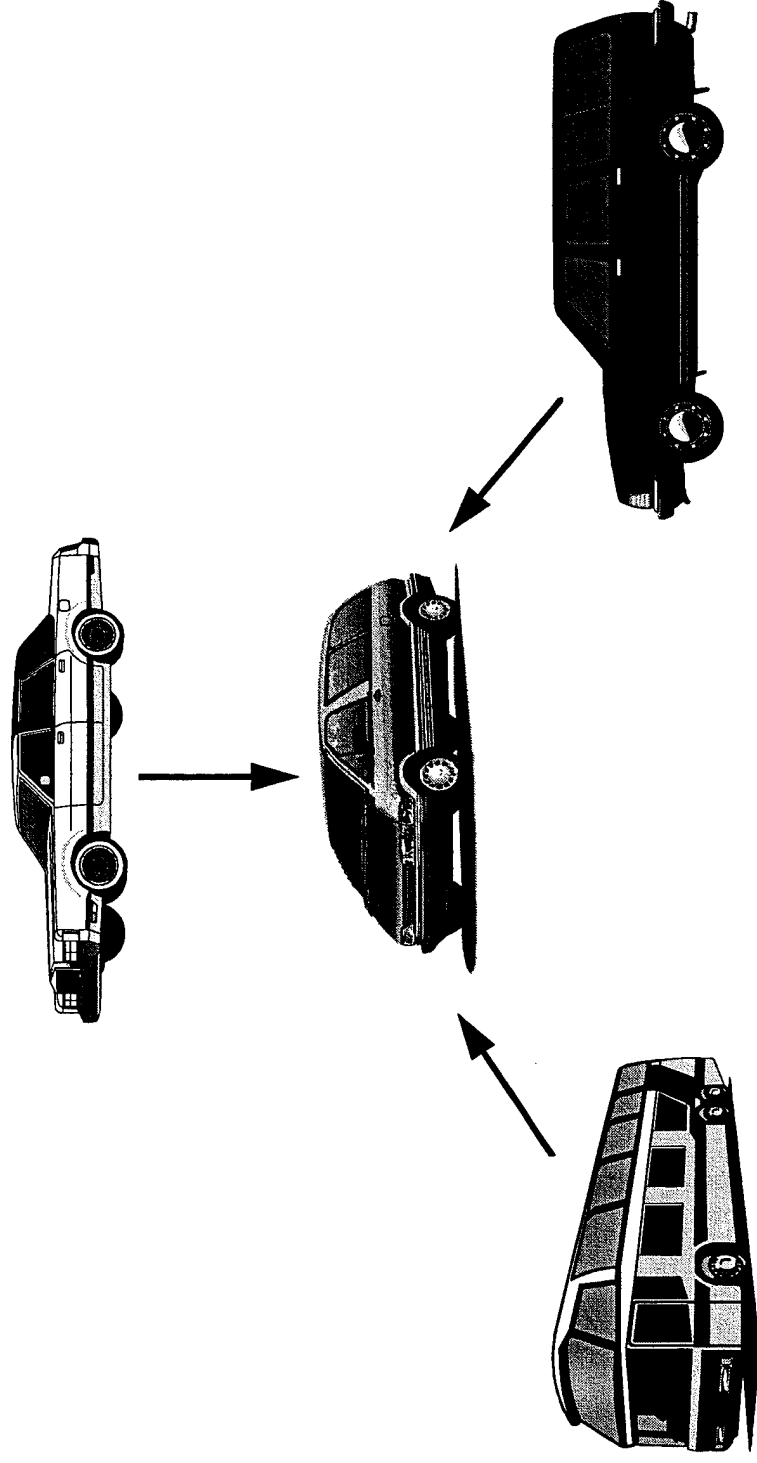
STUDENT INTERACTION

- Can you think of ways that industry has modularized products you use in your daily life?

OBJECTIVE

Students should be able to discuss, at a high level, what happens to the prescriptive model after it has been finalized.

5-4

Using a Prescriptive Model

- Create generic groups of functions using this model

DISCUSSION - Two Steps of Prescriptive Modeling

This slide shows the two major steps of prescriptive modeling. First of all, for example, new remote controls may require reuse of all functions that the descriptive model already illustrates. This means that the prescriptive model must bundle the descriptive model functions into modules. This model, of course, must be verified with the software developers before proceeding to design.

The next step is to enhance the descriptive model with possible new requirements and supporting functions.

STUDENT INTERACTIONS

- Can you enhance the prescriptive model for the remote control with features that it does not possess?
- How would you group these new features? Can the original prescriptive model accommodate them, or do you have to modify or extend it?

OBJECTIVE

Students should be able to enhance and update a descriptive model to accommodate new design requirements.

5-5

Two Steps of Prescriptive Modeling

- Bundle descriptive model functions into modules

- Enhance descriptive model with new requirements and supporting functions

5-5

SOFTWARE PRODUCTIVITY CONSORTIUM

DISCUSSION - Example

In this slide, the descriptive model illustrated in Unit 4 has been updated to accommodate a possible new requirement for the remote controller, which states that a combined remote controller is needed. The engineer, realizing an unstated need, has added a control media selection function.
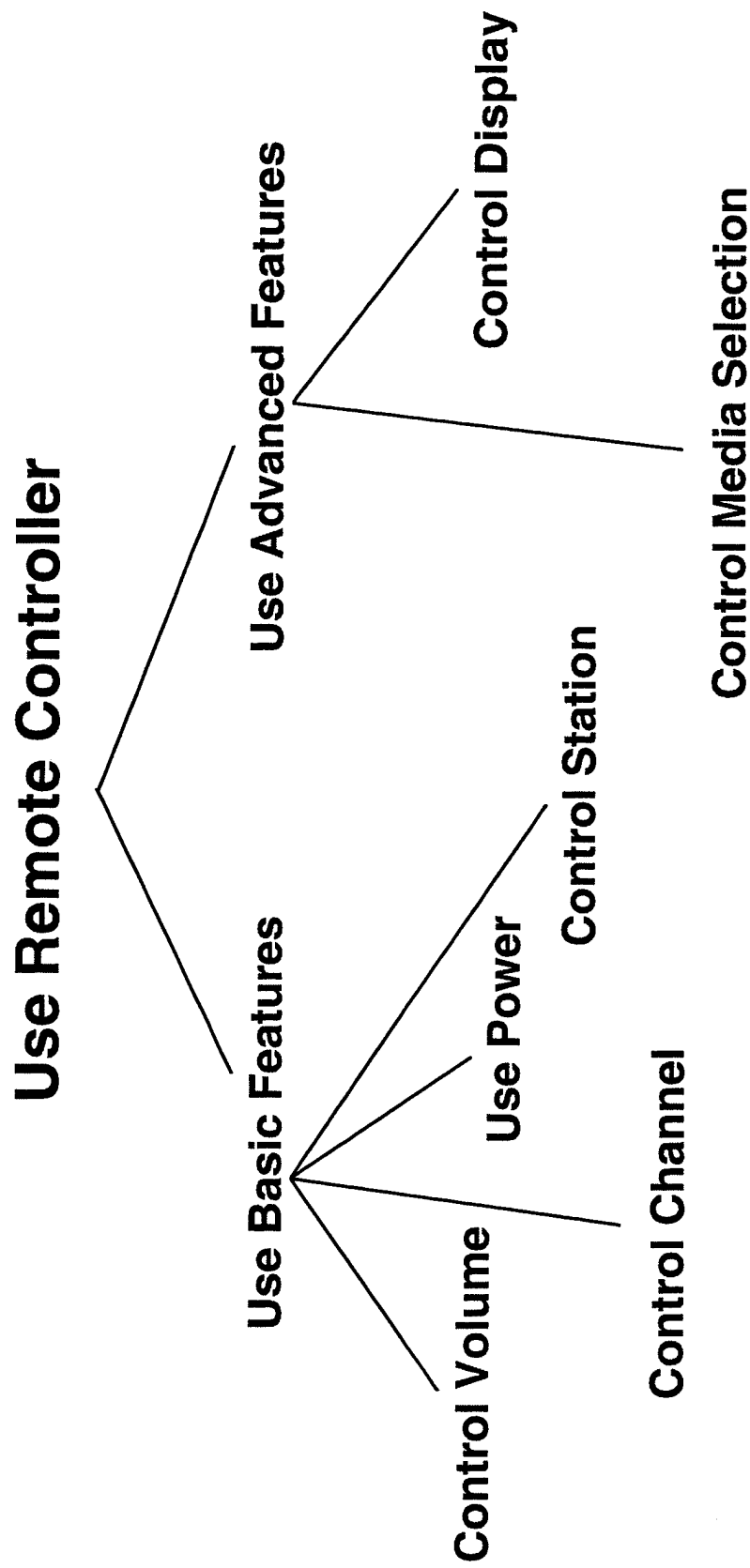
STUDENT INTERACTIONS

- Using some of the examples we have discussed in class, show what functions you would combine to create a model.
- Explain why you would combine these functions the way you did.

OBJECTIVE

Students should be able to understand why one would want to combine the various individual models.

5-6

Example

Use Remote Controller

Use Basic Features
- Control Volume
- Use Power
  - Control Station
  - Control Channel

Use Advanced Features
- Control Display
- Control Media Selection

SOFTWARE
PRODUCTIVITY
CONSORTIUM

DISCUSSION - Review

Now, let's review the development of a prescriptive model.

The prescriptive model, also known as the to-be model, is simply an extension of the descriptive model. The descriptive model is created based on old problems and their unique solutions. When a new system is to be designed, certain functions are likely to overlap with the old functions; thus, there is a need to create a descriptive model. The new system, however, will require updated functions, if not entirely new ones. The prescriptive model allows the engineer to estimate which updates or enhancements may be needed to accommodate future systems requirements.

Basically, a prescriptive model allows the domain analyst to model the best estimates for functions for new technology.

The slide shows how changing requirements have affected the design of cars from their early days to the present, and possibly how they will impact on how future cars will be designed.

STUDENT INTERACTION

What will the car of the future be? Ask this of students to end this part of the course. It will end the course on a very positive note (what's better than the future?).
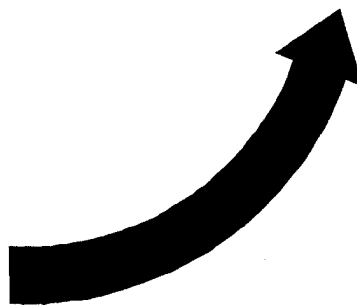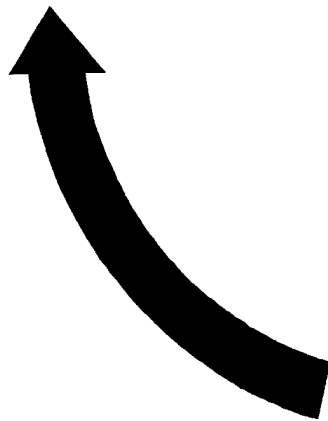
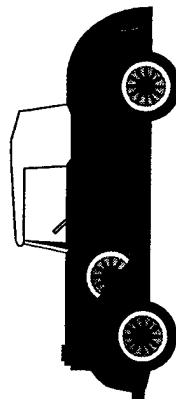OBJECTIVES

Students should be able to:

- State the reasons for developing a prescriptive model
- Describe how a prescriptive model is developed from an existing descriptive model

5-7

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Review

Car From the Past

Car in the Present

Car of the Future

?

DISCUSSION - Unit Summary

The prescriptive model, also known as the to-be model, is simply an extension of the descriptive model.

The descriptive model is created based on old problems and their unique solutions. When a new system is to be designed, certain functions are likely to overlap with the old functions; thus, there is a need to create a descriptive model.

The new system, however, will require more updated functions, if not entirely new ones. The prescriptive model allows the engineer to estimate which updates or enhancements may be needed to accommodate future systems requirements.

OBJECTIVES

Students will be able to:

- Identify the purpose of prescriptive models
- Describe the difference between descriptive and prescriptive models

Unit Summary

- **Prescriptive model:**

  — **Enhances the descriptive model with requirements for future systems within the defined domain**

  — **Hand-off model to domain design**

UNIT 5: PRESCRIPTIVE MODELS

Summary

In this unit, we discuss how the domain analyst enhances the descriptive model with features or functions that old solutions do not include. This model is referred to as the prescriptive model.

The prescriptive model, also known as the to-be model, is simply an extension of the descriptive model. The descriptive model is created based on old problems and their unique solutions. When a new system is to be designed, certain functions are likely to overlap with the old functions; thus, there is a need to create a descriptive model. The new system, however, will require updated functions, if not entirely new ones. The prescriptive model allows the engineer to estimate which updates or enhancements may be needed to accommodate future systems requirements.

Basically, a prescriptive model allows the domain analyst to model the best estimates for functions for new technology.

The prescriptive model draws upon the domain analysts' and software developers' knowledge and creativity to enhance the existing descriptive model for future requirements and functions. The idea behind modeling old requirements and functions for use in designing a new system is to provide the software developers with a perspective into past systems that they may not possess. Once they gain this knowledge, they can design a new system with features from the old system(s), and can design it for reuse in future systems.

Once completed, the prescriptive model serves as the input to domain design. Domain design uses the model to create or package functions into bundles or modularized functions. These functions are sufficiently generic so the software developers can pick and choose the functions they need for a new system.

Exercises

Exercise 1

For the common function called steering for the domains of cars, aircraft, and ships, generate a list of requirements and functions that you believe would enhance the existing steering system.

Exercise 2

For the domains of portable tape player, CD player, and AM/FM stereo, generate a list of requirements and functions for the common function you chose earlier, that you believe would enhance the existing sound system.

Exercise 3

Once the common descriptive model is built, incorporate new requirements by which the PEBOT is constrained, and enhance the existing domain model for future uses. Review documentation on the PEBOT and identify all information needed to control the PEBOTs joints. Whatever is missing from the descriptive model that the PEBOT needs becomes a new generic requirement for the domain model—the prescriptive model.

This model is now ready for the domain design phase of domain engineering.

Documentation is provided in the form of the Requirements Specification Sheet: Planetary Explorer Robot (PEBOT).

Teacher's Notes for Exercises

The following are notes for the teacher to apply to the exercises presented in this unit. The lists need not be comprehensive, but should be adequate to develop the students' understanding of the material.

Exercise 1

For the common function called steering for the domains of cars, aircraft, and ships, generate a list of requirements and functions that you believe would enhance the existing steering system.

Steering cars has not changed much at all since the inception of car design. Steering for aircraft, however, has changed with the creation of thrust vectoring and vertical takeoff. These two functions can be added to the descriptive model included in Exercise 1, Unit 4 under the steer left or steer right functions. Steering for ships also has not changed very much.

Exercise 2

For the domains of portable tape player, CD player, and AM/FM stereo, generate a list of requirements and functions for the common function you chose earlier that you believe would enhance the existing sound system.
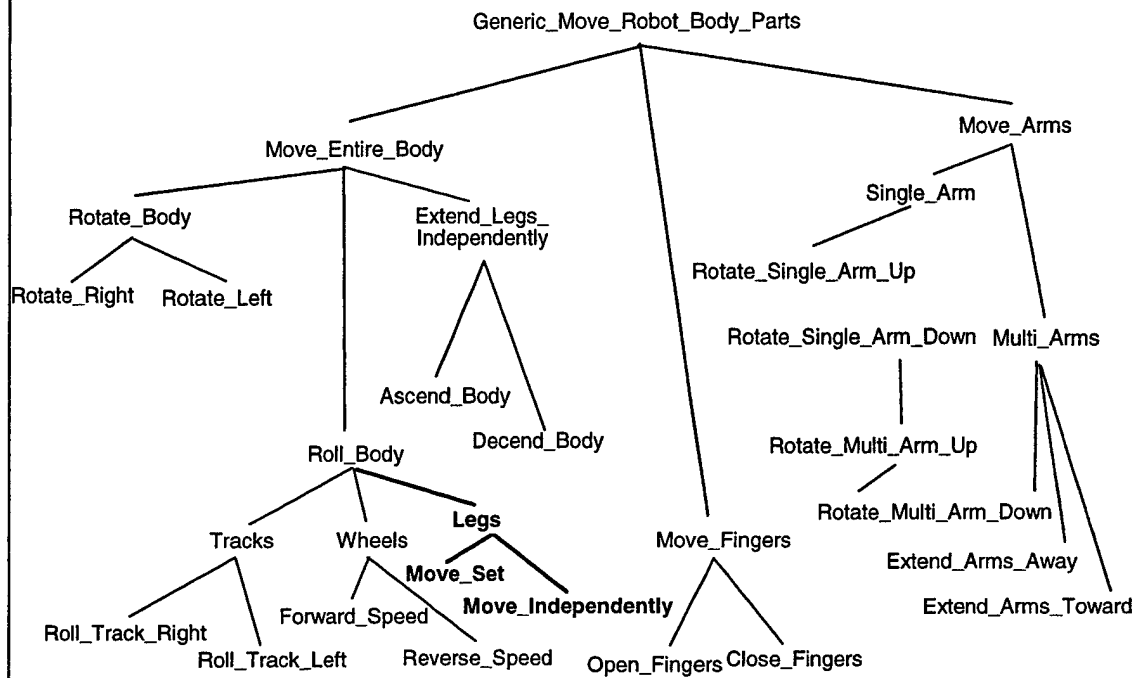
The functions that are graphically represented in the descriptive model in Exercise 2, Unit 4 is a basic model. Enhancements have been made by industry to each of the three audio components. Enhancements may include the following:

- Tape Player: dubbing function from one tape to another tape

- CD Player: programming function for programming select tracks from a number of CDs

- AM/FM stereo

Exercise 3

The following model depicts what might be considered enhancements to the robot mobility descriptive model, if PEBOT were considered the next generation robot. The bold lines and text show how the model differs from the descriptive model.

Prescriptive Model: Move_Robot_Body_Parts Function

Generic_Move_Robot_Body_Parts

Move_Entire_Body

Move_Arms

Rotate_Body

Extend_Legs_
Independently

Single_Arm

Rotate_Right   Rotate_Left

Rotate_Single_Arm_Up

Rotate_Single_Arm_Down   Multi_Arms

Ascend_Body

Decend_Body

Rotate_Multi_Arm_Up

Roll_Body

Rotate_Multi_Arm_Down

Legs   Move_Fingers

Tracks   Wheels

Extend_Arms_Away

Move_Set

Extend_Arms_Toward

Roll_Track_Right

Move_Independently

Forward_Speed

Roll_Track_Left   Reverse_Speed   Open_Fingers Close_Fingers

Requirements Specifications Sheet: Planetary Explorer Robot (PEBOT)

This requirements sheet presents the requirements and functional descriptions of the mobility system for a robot to be designed for planetary exploration.

Customer Need

Boom Industries must acquire a mobility system that is flexible enough to accommodate a robot that has no knowledge of the terrain on which it will operate.

Mission Statement

URW will develop a mobility system that can direct the PEBOT to move all its appendages and features in a undetermined environment. The PEBOT must be able to position itself without human intervention. The PEBOT must be able to position all its appendages without human intervention.

List of Requirements

REQ1:   The PEBOT must move independently and intelligently by being aware of any area within a radius of 10 feet.

REQ2:   The PEBOT must be able to recognize and retain (store) the path it follows to be able to return to a point that it has already visited.

REQ3:   The PEBOT must be able to move its body in three dimensions. It must be able to rotate, extend, retract, and pivot all appendages.

REQ4:   The PEBOT must be able to gather objects and test them for various chemical and elemental compositions.

REQ5:   The PEBOT must be able to store some objects and return them to the point of origin.

DISCUSSION - Unit 6: Introduction to Domain Design, Problems and Solutions

Now we begin a new phase of megaprogramming, domain design. We will learn to consider providing solutions to problems we described during domain analysis. We are not just interested in determining a single, specific solution to a problem. Rather, we are concerned with recognizing that a set of solutions can address one or more similar problems. We will come to understand the importance of keeping the problem and solution separate so we can apply the most effective approach to a a specific problem.

In this unit we will discuss the need for domain design and domain design's relationship to domain analysis. In unit 7 (architectures), we will talk about organizing our solutions so that they are easily adapted to a set of problems. In unit 8 (writing reusable software), we will talk about how to write software that can be used to solve many different problems based on the solutions we've designed.

OBJECTIVES FOR THE UNIT

Students should be able to:

- Understand the relationship among megaprogramming, domain analysis, and domain design
- Describe the relationship between problems and solutions

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Unit 6: Introduction to Domain Design Problems and Solutions

DISCUSSION - Domain Design

The application engineer uses the products of domain design to acquire knowledge from past similar solutions. Domain design covers the following topics:

- Differences between problems and solutions
- Architectures (in particular, software architectures), which are the most important product of domain design phase
- How to write reusable software code that supports the software architecture
- How to build a software application using a software architecture

The following slides on problems and solutions illustrate the concepts of multiple problems with the same solution and multiple solutions for the same problem. This sets the context for a discuss of architectures, which provide a single solution for multiple problems. This represents a shift in perspective from a problem orientation to a solution orientation; this shift generally takes place with the transition from domain analysis to domain design.
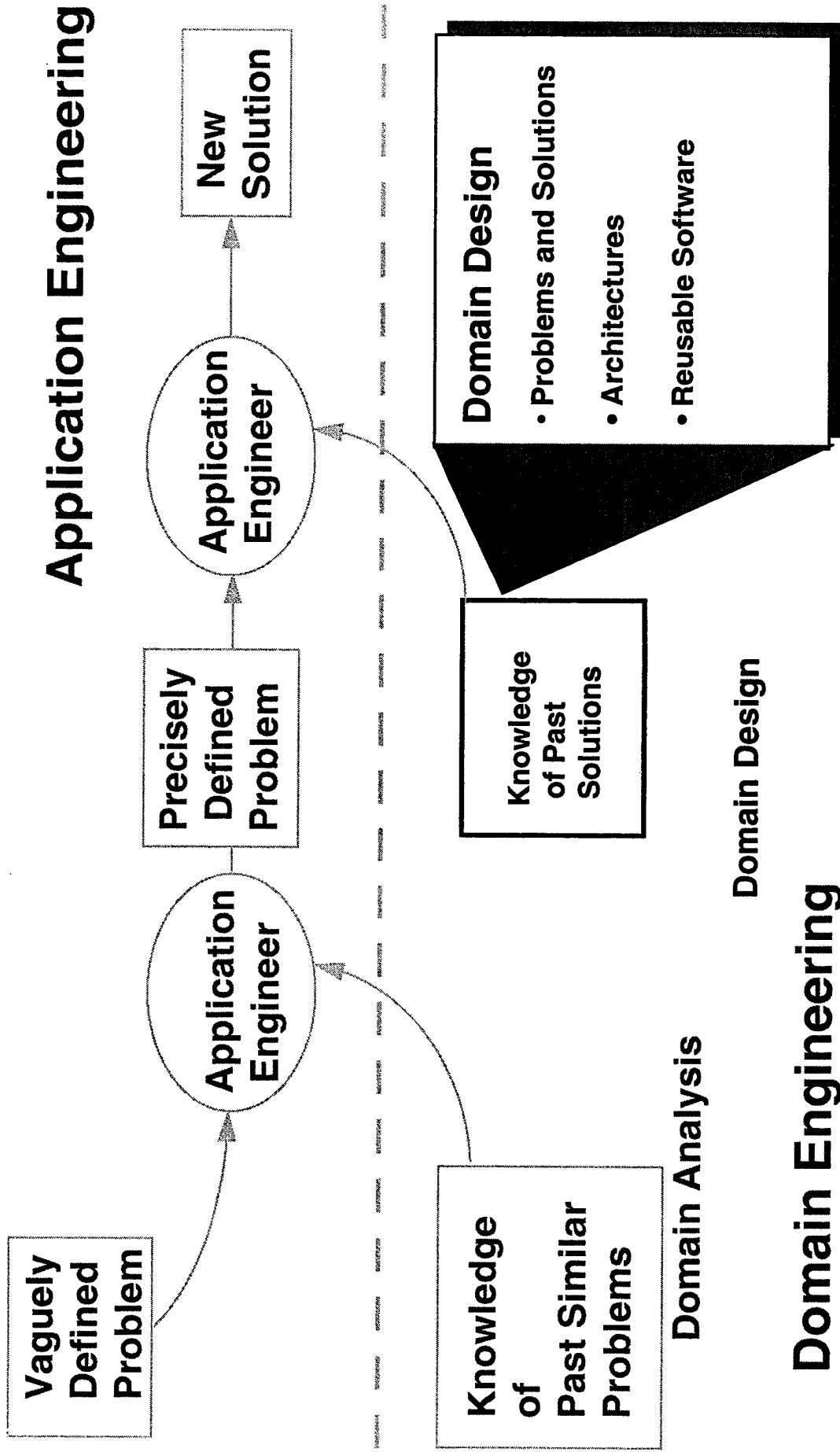
After completing the architecture unit, we will discuss how to write software whose purpose is not to solve just one problem. We will discuss writing reusable software to solve many problems. Finally, we will discuss briefly how one can use an architecture to build software applications.

OBJECTIVES

Students should be able to:

- Define the topics of domain design
- Describe why one begins by looking at problems and solutions

6-2

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Domain Design

Application Engineering

Vaguely
Defined
Problem → Application Engineer → Precisely
Defined
Problem → Application Engineer → New
Solution

Knowledge
of
Past Similar
Problems

Knowledge
of Past
Solutions

Domain Design
- Problems and Solutions
- Architectures
- Reusable Software

Domain Analysis   Domain Design

Domain Engineering

6-2

SOFTWARE
PRODUCTIVITY
CONSORTIUM

DISCUSSION - Problems and Solutions

We will now revisit some concepts with which you are already familiar: problems and solutions. Let's first look at problems. Similar kinds of problems are experienced by many people. For example, how many people in this room have had a common cold and fever? Some problems are experienced by specific groups of people. For example, all car owners share the similar kinds of problems: having enough gasoline, tire punctures, engine malfunctions, dead batteries, etc.

Now, let's look at solutions. Why do you think there are so many different products on the market? Each product or service is intended to solve one or more problems for the consumers. For example, cold medicines alleviate common colds, while car mechanics try to solve automotive problems.

Next, we will look at the combination of problems and solutions. Multiple problems can generally be solved by a single solution, and multiple solutions may exist for a single problem.

STUDENT INTERACTION

How many people do you know who actually build a car from scratch for their personal use?

OBJECTIVES

Students should be able to:

- Recognize that many problems are common to many people
- Determine that common solutions exist for many problems, and unique solutions may have undesirable characteristics

Problems and Solutions

- **Problems**
  - **Cold**
  - **Fever**

  - **Need gasoline**
  - **Flat tire**
  - **Dead battery**
  - **Engine malfunction**

- **Solutions**
  - **Cold medicines**

  - **Auto mechanics**

DISCUSSION - Many Problems, One Solution

A single solution may solve multiple problems.

The house in this slide is a good example. A house provides solutions to many needs. It can be used to rest, to work, to lead a life in an atmosphere unaffected by such external environmental factors as heat, cold, or rain.

A solution is designed to solve a particular problem or set of problems. A solution may also come with some options to address a range of problems.

A house has certain amenities: a kitchen, fireplace, air conditioning, garage, and bedrooms. Some of these amenities, e.g., a fireplace or garage, may be optional.

When you build a solution, more people can use your solution (product or service) if it solves either multiple problems or problems frequently experienced by many people.

STUDENT INTERACTIONS

- Describe three more examples of single solutions that can be used to solve multiple problems.
- How many different activities take place in your house or apartment?
- Does your family use a car just to drive you to school?

OBJECTIVES

Students should be able to:

- Recognize that a solution to one problem may be applied to other problems
- State the advantages of generating a solution that applies to multiple problems

6-4

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Many Problems, One Solution

Need a place to think

Need a place to relax

Need a place to work

Need a place to eat

A house!

Need a place to rest
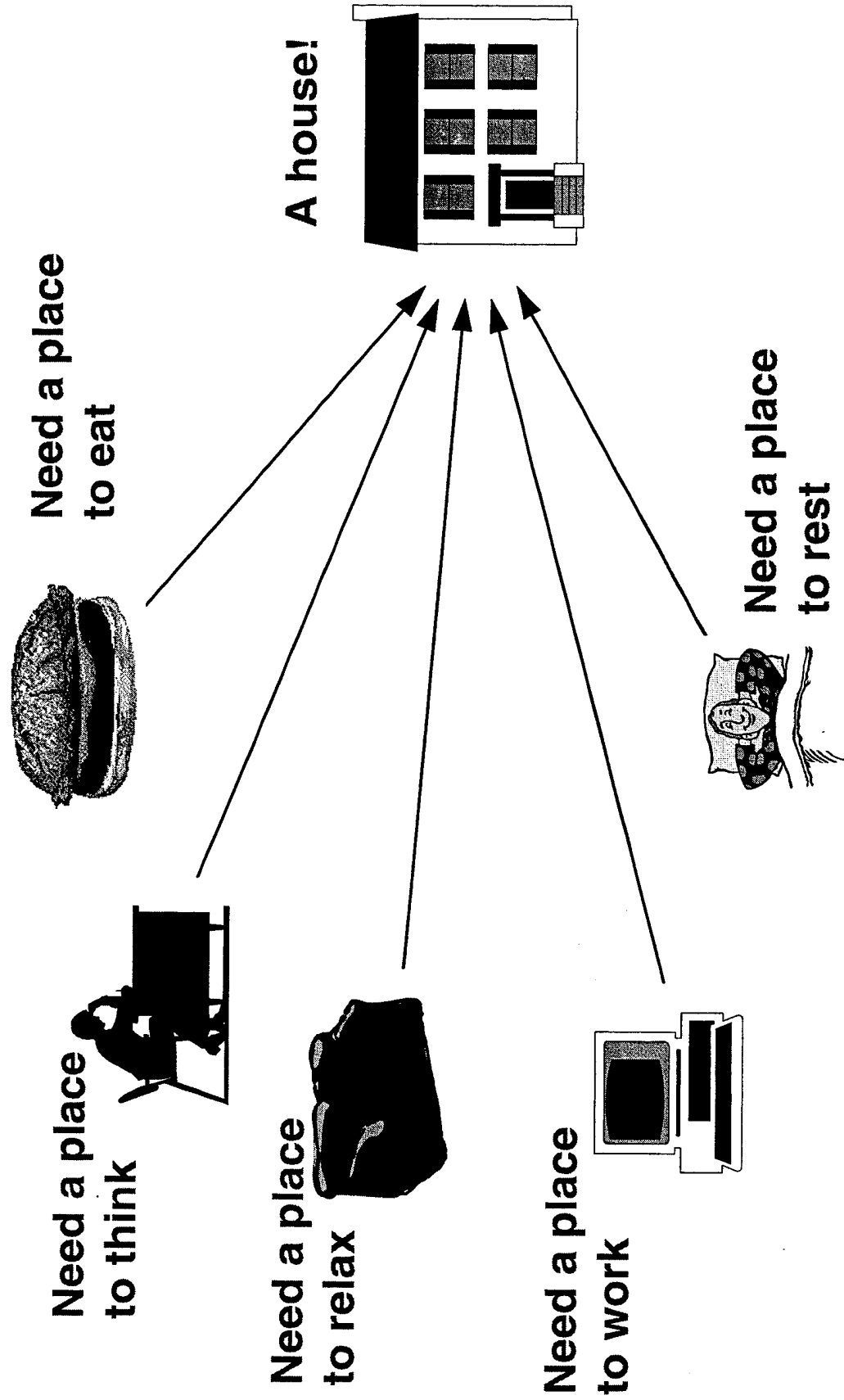
DISCUSSION - One Problem, Many Solutions

One problem can have many solutions.

For example, this slide shows that if you have to get from one place to another, you can walk, ride a horse, use a car, take a plane, etc. In a second situation, a farmer who has to harvest corn can harvest it manually himself, employ somebody else to do it, or rent or own a robot to harvest the corn.

The precise solution for a given problem depends on several factors, including economics, technology, preferences, and others.

STUDENT INTERACTION

Develop three other examples of one problem with many possible solutions.

OBJECTIVES

Students should be able to:

- State why one problem may have many solutions
- Explain why identifying more than one solution to a problem is important

SOFTWARE
PRODUCTIVITY
CONSORTIUM

One Problem, Many Solutions

Sailing

Flying

Driving

Riding

Biking

Trucking

Need transportation?

SOFTWARE PRODUCTIVITY CONSORTIUM

6-5
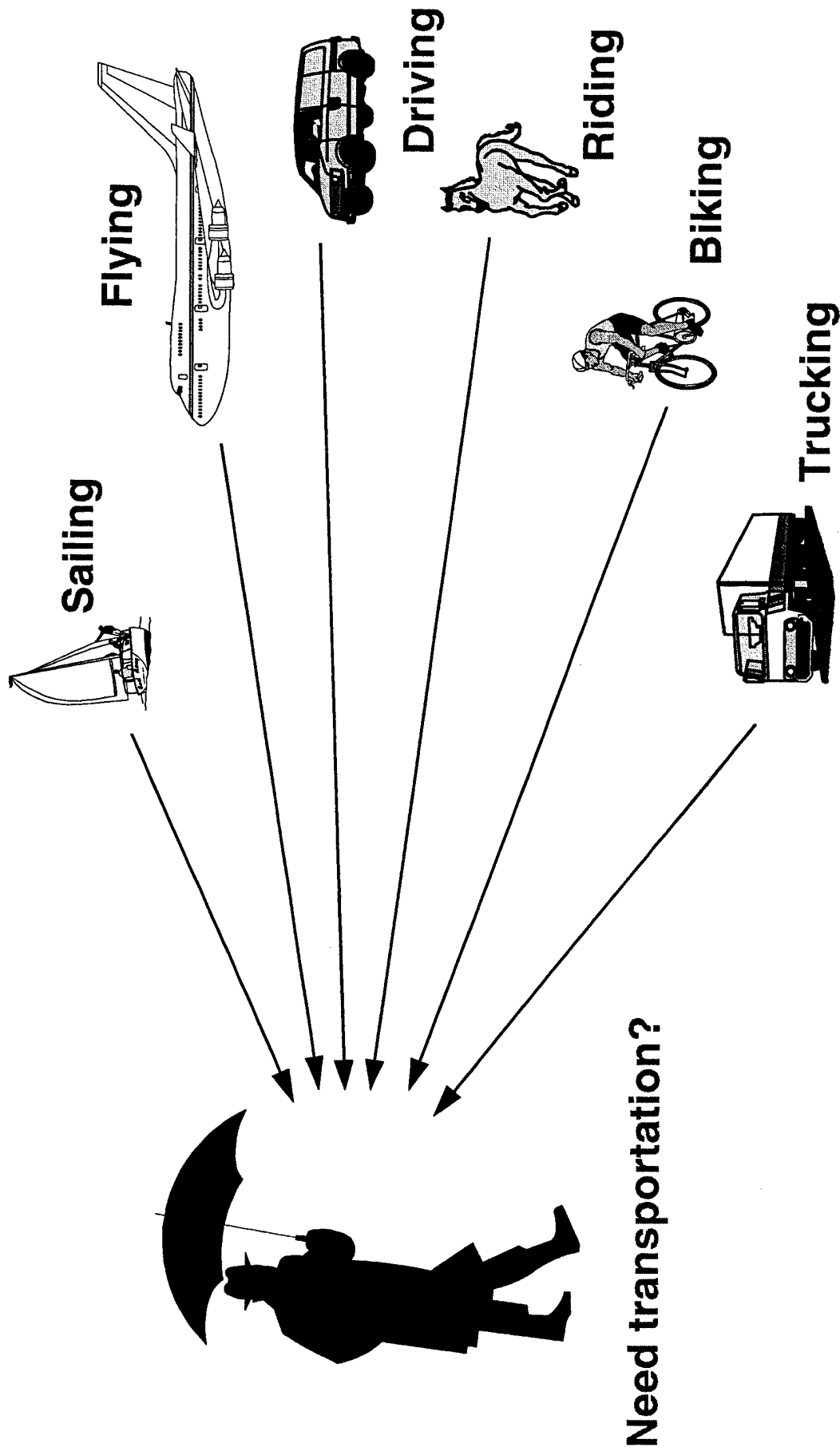
DISCUSSION - Unit Summary

Problem Space versus Solution Space

- Problem: To lead a life in a cozy atmosphere unaffected by external environmental factors (heat, cold, rain, etc.).
- Solution: A house with certain amenities (kitchen, heater, air conditioning, bedrooms, etc.). Note that some of these amenities may be optional (such as heater, if you lived in the Virgin Islands).
- Problem: To get from one place to another in a fast, safe, and cost-effective manner.
- Solution: A vehicle, such as a car or a truck. Again, cars are equipped with certain amenities, such as air conditioning, cruise control, four-wheel drive, and antilock brakes, some of which are optional.

Some problems are commonly encountered by millions of people over and over again. Some of these problems transcend generations and some seem universal to humankind (for example, to minimize adverse effects of weather on our lives).

Domain analysis uses knowledge of past similar problems to define precisely a new problem. Domain design uses knowledge of past similar solutions to develop a new solution to this new problem. Our focus here is no longer on defining the problem, but on generating a solution. Specifically, generating a reusable solution is the goal.

STUDENT INTERACTIONS

- Are the stated solutions to these problems reasonable?
- Could the solutions be used in solving other problems?

OBJECTIVES

Students should be able to:

- Describe the relationship between domain analysis and domain design
- Explain the present focus on solutions

6-6

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Unit Summary

- Domain analysis uses knowledge of past similar problems to define precisely a new problem.

- Domain design uses knowledge of past similar solutions to develop a new solution to this new problem.

- Many problems can be solved with one solution.

- One problem can have many solutions.

- Domain analysis has a problem orientation; domain design has a solution orientation.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

UNIT 6: INTRODUCTION TO DOMAIN DESIGN

Summary

This unit introduces a new phase of megaprogramming, domain design. The application engineer uses the products of domain design to acquire knowledge of past similar solutions. domain design covers the following topics:

- Differences between problems and solutions

- Architectures (in particular, software architectures), which is the most important product of the domain design phase

- How to write reusable software code that supports the software architecture

- How to build a software application using a software architecture

This process involves some concepts with which you are already familiar: problems and solutions. Let's first look at problems. Similar kinds of problems are experienced by many people. Some problems are experienced by specific groups of people.

Next, let's look at solutions. There are so many different products on the market because each product or service is intended to solve one or more problems for the consumers.

Finally, let's consider the combination of problems and solutions. Multiple problems can generally be solved by a single solution, and multiple solutions may exist for a single problem.

A single solution may solve multiple problems. Each solution is designed to solve a particular problem or set of problems. A solution may also come with some options to address a range of problems. When you build a solution, more people can use your solution (product or service) if it solves either multiple problems or problems frequently experienced by many people.

One problem can have many solutions. The precise solution for a given problem depends on several factors (economics, technology, preferences, etc.).

Some problems are commonly encountered by millions of people over and over again. Some of these problems transcend generations, and some seem universal to humankind (for example, to minimize the adverse effects of weather on our lives).

Domain analysis uses knowledge of past similar problems to define precisely a new problem. Domain design uses knowledge of past similar solutions to develop a new solution to this new problem. Our focus here is no longer on defining the problem, but on generating a solution. Specifically, generating a reusable solution is the goal.

Exercises

Identifying Problems

Identify and write down the problem you think the following solutions would solve:

1. Chair, bench, couch, floor, boulder, steps

2. Cup, bowl, plastic bag, spoon, palm of hand, ladle

Identifying Solutions

Identify and write down solutions you think would solve the following problems:

1. Turning someone down who asks you for a date

2. What to do with the money you got for your birthday

3. What to eat for breakfast

Matching Problems and Solutions

Match the following problems and solutions:

1. Problem: Get a message to your friend in Asia

2. Problem: Record a "help" note

3. Problem: Hold several pieces of paper together

    a) Solution: staple, tape, gum, glue, paper clip, rubber band

    b) Solution: telegram, Fed-X, United States Postal Service, UPS, Internet

    c) Solution: lipstick, pen, pencil, magic marker, highlighter

Teacher's Notes for Exercises

Identifying Problems

Identify and write down the problem you think the following solutions would solve:

1. Chair, bench, couch, floor, boulder, steps

   - *A place to sit down*

2. Cup, bowl, plastic bag, spoon, palm of hand, ladle

   - *Something in which to transport a small amount of liquid*

Identifying Solutions

Identify and write down solutions you think would solve the following problems:

1. Turning someone down who asks you for a date

   - *Turn and run, make up an excuse, explain it gently*

2. What to do with the money you got for your birthday

   - *Buy something, donate it to a charity, save it, give it to a family member, repay a debt*

3. What to eat for breakfast

   - *Cereal, eggs and toast, leftovers*

Matching Problems and Solutions

Match the following problems and solutions:

1. Problem: Get a message to your friend in Asia

   a) *Solution(5): telegram, Fed-X, United States Post Office, UPS, Internet*

2. Problem: Record a "help" note

   b) *Solution(6): lipstick, pen, pencil, marking pen, highlighter*

3. Problem: Hold several pieces of paper together

   c) *Solution(4): staple, tape, gum, glue, paper clip, rubber band*

DISCUSSION - Unit 7: Software Architectures

In this unit, we will discuss the most important product of domain design: software architectures.

Domain designers take the information gathered and organized during domain analysis and create a formal solution. As with domain engineering in general, domain design is not a clear-cut, step-by-step process. It is a set of iterative, high-level activities. In general, domain design consists of designing a domain architecture and developing guidelines for the use of this architecture.

This unit presents an extensive introduction to software architectures.

OBJECTIVES FOR THE UNIT

Students should be able to explain the role of architectures in domain design.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Unit 7: Software Architectures

7-1

DISCUSSION - Examples of Architectures

This slide presents some real-world examples of architectures: cameras and automobiles. We will see later that software architectures are very similar to these examples.

Architectures are nothing but a well-established way of building solutions to certain problems. Every camera we see has a shape similar to every other camera. Every car we see looks similar and acts in similar ways to other cars. But do all the cameras and cars look and function exactly the same? There are definite variations. We can all agree, however, that they have more commonalties than differences. Very rarely will we mistake a car for a bus, or a camera for a blender! Thus, we can say that all cars are based on a common car architecture and that all cameras are based on a common camera architecture.

Why do you think cars look the way they do and cameras look the way they do? They each have a common solution "template" for a specific problem or a given set of problems.

STUDENT INTERACTIONS

- What jumps to your mind when we talk about architectures? homes? office buildings?
- Why do you think architectures exist?

OBJECTIVES

Students should be able to:

- Describe the concept of architectures
- Explain why architectures are used in domain design

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Examples of Architectures

Windows

Roof
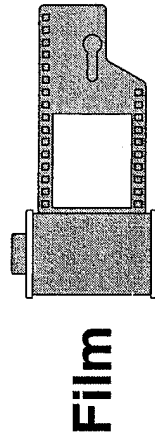
Engine

Doors Wheels

Automobile

Film

Viewfinder

Shutter Release Button

Film-advance Lever

Lens

Camera Case
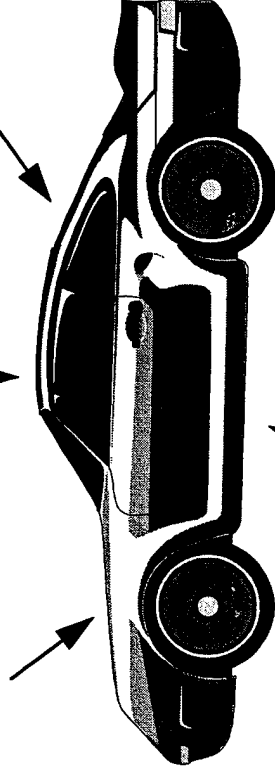
Camera

DISCUSSION - Three Attributes of an Architecture

Now that we have looked at some examples of architectures, let's look at them more closely. At a high level, every architecture has three attributes or components: style, parts, and assembly instructions.

- An architecture has various styles. Houses may be Gothic, Victorian, ranch, split-level, or log cabin; robots may be egg-shaped (R2D2), humanoid, or spider; and software may be distributed, event-driven, peer-to-peer, or host-based.

- An architecture is made up of parts (or elements). These parts have specifications, such as minimum and maximum values or ranges of operation. These parts also serve certain functions. A house has bathrooms, bedrooms, and a kitchen; a robot has a scanner, image analyzer, communicator, and power-supply; and software can have database, input/output, and word processing functions.

- An architecture has assembly instructions or composition rules, which are sometimes called generation procedures. Instructions for a house tell you where to put the garage and front and back doors; for a robot, instructions tell you what kind of scanners will work with the image analysis and power requirements of the scanner (6v–11v); and software instructions explain what data types the database can manipulate or how much memory is needed to run it.

STUDENT INTERACTIONS

- Do you think building software is as easy as building the entertainment center?
- Why are instructions important in a word processing package?

OBJECTIVES

Students should be able to:

- List the three major attributes of an architecture
- List three styles of a particular architecture
- List three parts of a particular architecture

7-3

SOFTWARE
PRODUCTIVITY
CONSORTIUM

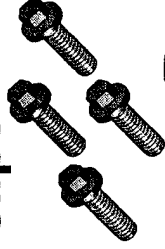# Three Attributes of an Architecture

Style

Parts
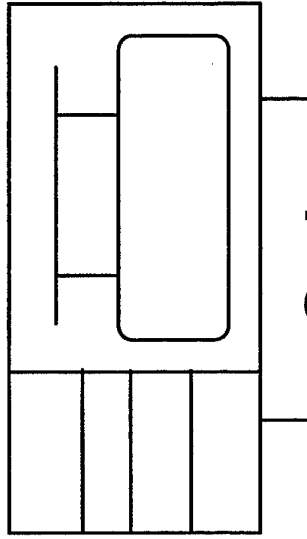
Screws
Plastic clips

Instructions

Attach shelf (A) to side (B). Place plastic clip between shelf - side joint, and then connect with screw (C).

Entertainment Center

7-3

SOFTWARE
PRODUCTIVITY
CONSORTIUM

DISCUSSION - Architecture Styles

This slide illustrates architecture styles. We have seen in earlier slides that a given architecture has certain styles associated with it. When looking at robots, we can see certain styles of robots: R2D2 (egg-shaped), humanoid (shaped like a human), or spider (with a number of legs).

Why do you think architectures have different styles? Do you think R2D2 would do well in searching for lost people in a forest? Can you picture R2D2 trying to climb little rocks and jump across streams? At the same time, do you think a spider robot that is designed to travel over rough terrain would be useful on a small farm to pick tomatoes? Even if you could use a spider robot, it would be like using a bulldozer to dig up your summer garden.

Different architecture styles exist to solve different kinds of problems within the same problem area. Styles are just variations of the same basic architecture. Remember, a robot is still called a robot, whether it is egg-shaped, humanoid, or spider-shaped.

Even though you see different architecture styles, they all share many commonalties with each other. Each of our robots has to look around and analyze the images (tomatoes, humans, or streams) they see. They all have to decide what is the shortest and safest way to get from where they are to their next destination. They all have to move their different body parts (legs, arms, and head).

STUDENT INTERACTIONS

- Why do you think most houses in the northern parts of the United State and Canada have sloped roofs? You hardly see buildings with flat roofs. Why is that?
- Why do you think houses built close to rivers, as well as houses in low-lying areas, have raised floors?
- Why do you think most cars sold in Africa and India do not have heaters as standard equipment?
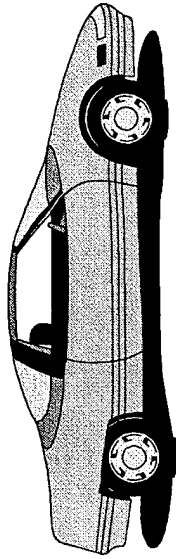
OBJECTIVE

Students should be able to explain the rationale for the existence of styles within architectures.
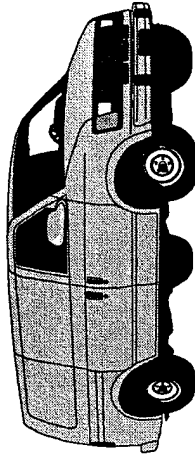
SOFTWARE
PRODUCTIVITY
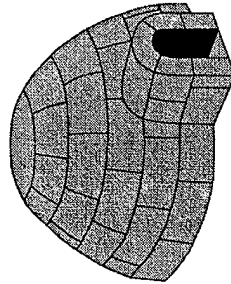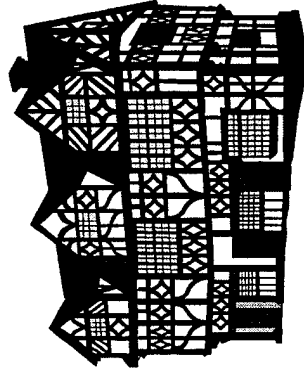CONSORTIUM

Architecture Styles
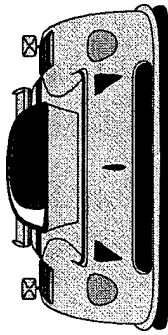
Sports

Family

Minivan

Ranch

Igloo

Tudor

Humanoid

R2D2

Spider

SOFTWARE PRODUCTIVITY CONSORTIUM
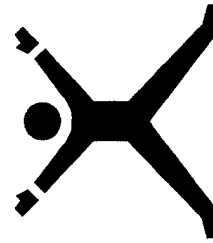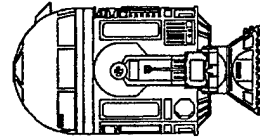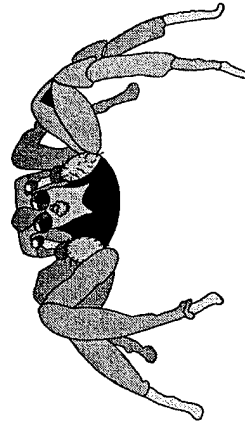
DISCUSSION - Architecture Styles Evolve

Just as cars, robots, and houses have architecture styles, the software industry is defining software architecture styles that suit specific needs. Since the software industry is a young industry, its architecture styles are still evolving.
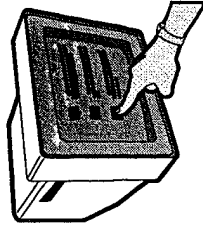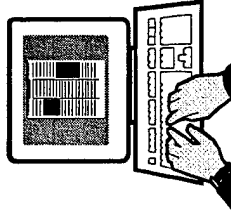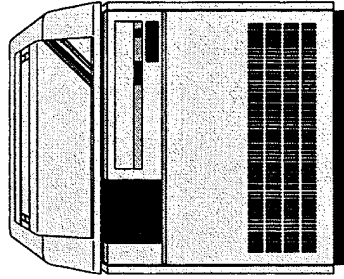
STUDENT INTERACTIONS

How will houses evolve? cameras and taking pictures? computers and software?

OBJECTIVE

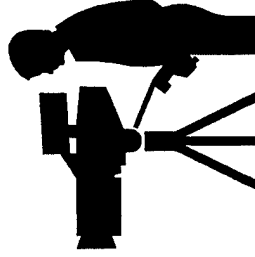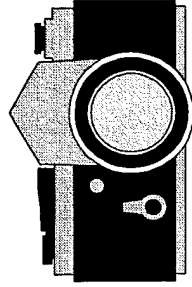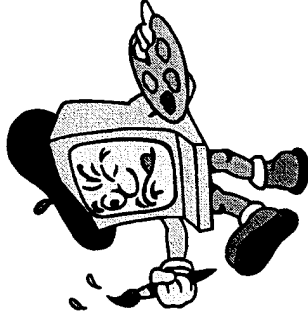Students should be able to recognize that architecture styles are continually evolving.
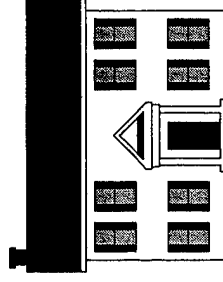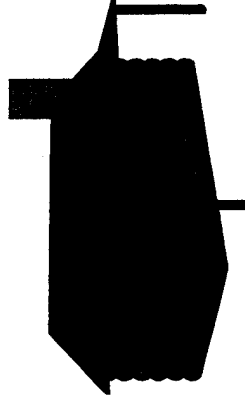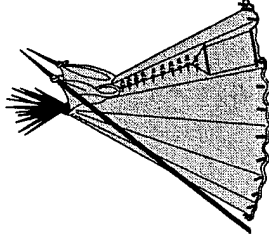
7-5

Architecture Styles Evolve

Houses:

1,000 years old

Pictures:

100 years old

Software:

50 years old

SOFTWARE
PRODUCTIVITY
CONSORTIUM

7-5

DISCUSSION - Styles of Software Architecture

Just as homes, cars, and cameras have architecture styles, software also has styles. Software styles have evolved over time to meet various needs, and they will continue to evolve since the software industry is so young. Let us look at couple of software architecture styles and see why they exist.

- Communicating processes and event systems architectures were developed to support independent software systems or components to work together. For example, an airline reservation software system is a suitable candidate for a communicating processes architecture due to the broad geographic dispersion of the personnel (reservation clerks, ticket agents, and passengers) who need to use the software system.

- Transactional and blackboard software architectures are useful in building software systems that have a lot of data to be shared by a number of users and/or software systems. In these architectures, the data is stored centrally and retrieved and updated by various users.

Typically, a software architecture may embody more than one architecture style. This is similar to a house that is colonial in style, but that has a modern kitchen.

For example, at a very high level, a software system may resemble a communicating processes architecture, and then at lower level, each "box" may resemble a different architecture style.

STUDENT INTERACTION

Discuss why software and houses are similar when looking at architectural styles.

OBJECTIVE

Students should be able to understand that architecture styles apply to software just as they apply to physical objects.

Styles of Software Architecture
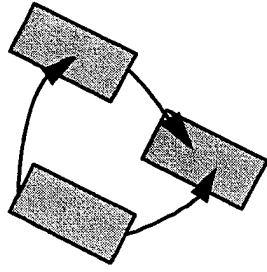
Communicating Processes
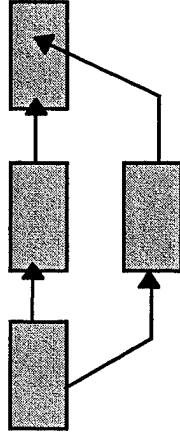
Batch Sequential
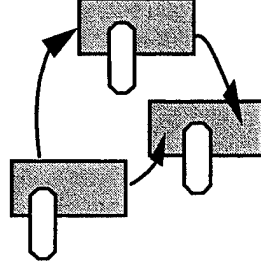
Object Oriented
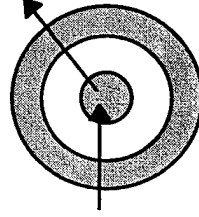
Layered

Event Systems
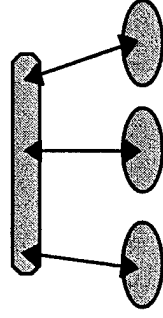
Transactional and Blackboard
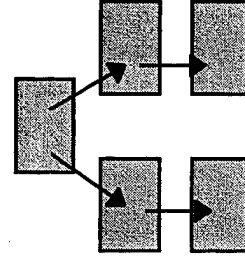
Main Programs and Subroutines
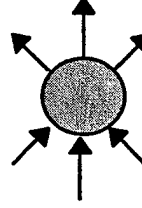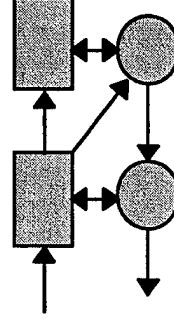
Rule-Based Systems Interpreters

DISCUSSION - Parts and Interconnectivity

On the previous slide, we saw that each architecture is a collection of parts. For each style of architecture, we can see clearly the different parts that make up that architecture. A robot has eyes, arms, legs, and torso. An entertainment center has boards, glass, nuts, and bolts.

Since it is our job to help others build their own systems, we must provide a complete description of each part of the architecture. Suppose we were given only a picture of a house (an architecture style) and asked to build it ourselves. We could go to the lumber yard and buy the necessary parts. But first we would have to figure out the type of materials and their characteristics (e.g., size of windows), how many, where to put the walls, and so on. This tends to get complicated. Even if we could figure all this out, we would still have to decide the order and manner in which to assemble everything.

To make using architectures simple, we must provide a detailed description of each part. For our robot, we should describe how long the legs should be, what kind of material should be used in cold temperatures (for use in the arctic or in the exploration of cold planets), or in very hot temperatures (for exploration of very warm planets). We should provide information about where to find the materials for the legs. If possible, estimates of the cost of the materials and the time for completion should be given. This description should include every detail that is necessary for someone to build the robot leg without outside help.

A detailed description should be given for each and every part of the architecture. These descriptions will help not only in building the parts from scratch (to write brand new software code, in the case of software architecture), but also in evaluating existing pieces to see if they can be used with the current architecture. In the case of software architecture, we can see if the existing software code can be used in the new architecture.

STUDENT INTERACTIONS

- Have you seen descriptions of pieces of software code? If so, elaborate.
- If you were given detailed descriptions of each part of the entertainment center, how easy would it be for you to find the needed parts in the local stores?
- If you were given detailed descriptions for different software parts, how easy do you think it would be to find the needed parts? Where would you look?

OBJECTIVE

Students should be able to explain the need to provide detailed descriptions of each part that makes up an architecture.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Parts and Interconnectivity

Parts of a Robot

Head

Arm
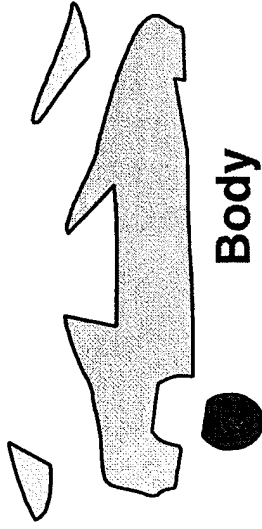
Foot

Parts of a House

Roof

Chimney

Window

Porch Roof

Door Frame

Door

Parts of a Car

Roof

Front Windshield

Rear Windshield

Body

Wheels

7-7
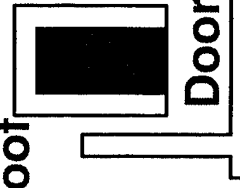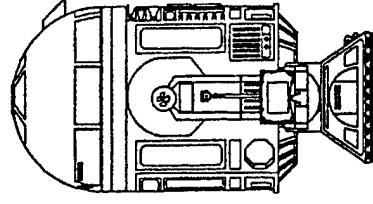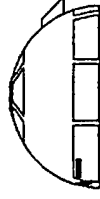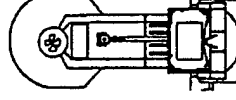
DISCUSSION - Software Parts of a Robot

A software architecture defines the software parts and how they work together. Even simple software systems such as a word processor have different software parts: keyboard interface, spellchecker, printer interface, and display manager.

The slide shows the different parts of a software architecture for a robot.

When building a software system, it is essential to know the different parts of the system. We should also know the capabilities of the different parts. A software application developer can use this information to determine whether parts are adequate or inadequate for the purposes at hand. For example, when selecting an image analyzer for an agricultural robot, the developer must make sure it can recognize vegetables from lumps of clay to be useful to the farmer.

Thus, it is very important that when you develop a software component, you must state clearly the capabilities and limitations of the component so others can evaluate it easily.

STUDENT INTERACTIONS

- Select a type of software and describe the components of that software.
- Can you describe the capabilities of each of the software components?

OBJECTIVE

Students should be able to understand the need to identify software parts and the capabilities of the parts.

7-8

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Software Parts of a Robot

```
                    Communicator
   Image                               Finger
   Analyzer                            Controller
          \       Central       /
           \      Processing   /
            \                 /
             Leg        Torso       Arm
             Controller Controller  Controller
```

Image Analyzer → Central Processing

Central Processing ↔ Communicator

Central Processing ↔ Finger Controller

Central Processing ↔ Torso Controller

Central Processing ↔ Arm Controller

Central Processing ↔ Leg Controller
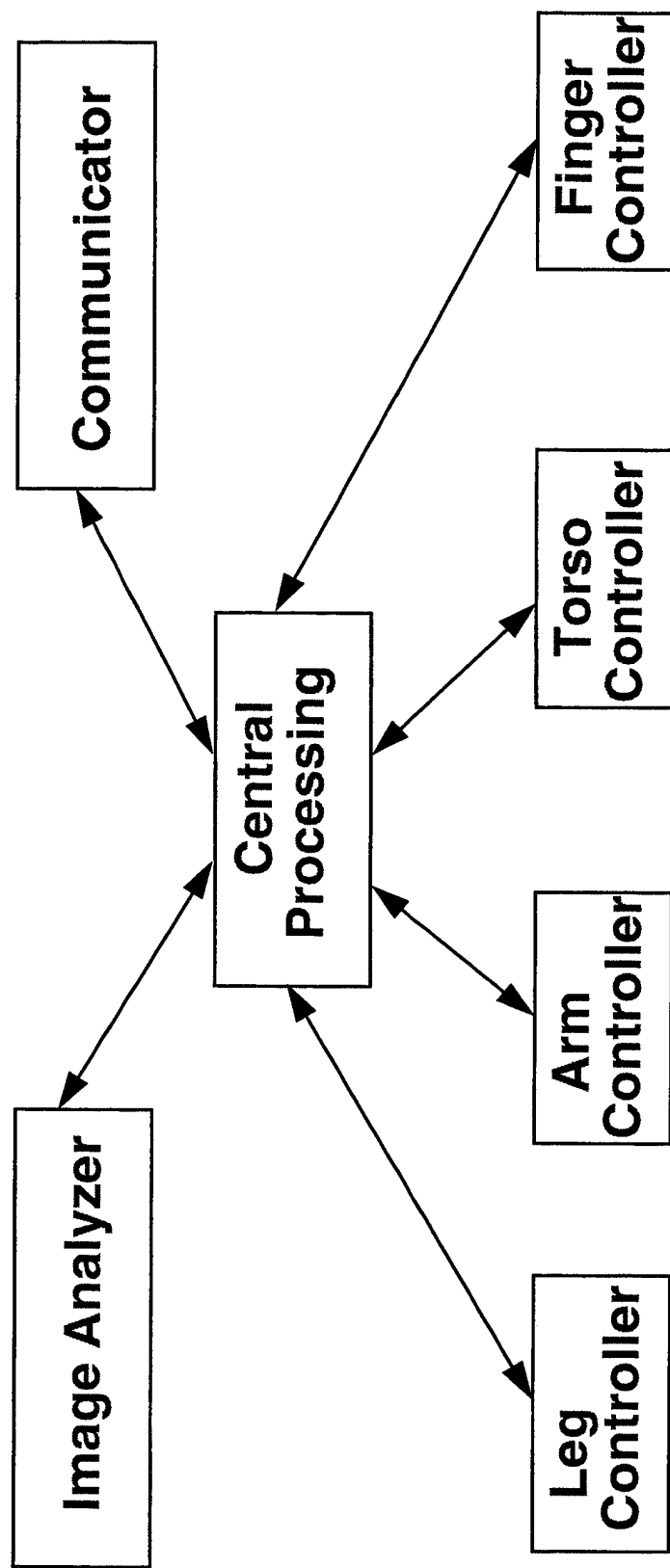
DISCUSSION - Examples of Instructions

We now understand that an architecture is made up of several different parts. It is very important to note that these parts are connected to each other as defined by the architecture. The interconnections between the parts are what is of interest to our customers. What good is a pile of brand new car parts sitting in our garage, if they are not connected properly to each other?

If we are going to help others build something, it is not enough just to provide them with an architecture style and a list of parts. Assume that you bought an entertainment center at the local store, and they gave you the style of the entertainment center, a picture of how it looks, and all the parts, but forgot to give you the instructions. It would be very difficult to build the entertainment center. In a way, assembly instructions provide the "glue" that is necessary to put all the parts together.

Obviously, the assembly instructions be prepared for the chosen architecture style. Assembly instructions for an egg-shaped robot will not be helpful when you are trying to build a spider-like robot.

We know that each architecture has some essential parts (or core parts) and some nonessential parts. Let's suppose that a given house architecture at the least should have a bedroom, living room, kitchen, and bathroom. These, then, are the core, or essential parts, of the house. The given architecture can also accommodate a deck in the back and a garage. With this architecture, we can build several different homes.

The assembly instructions must address the various ways of putting together the parts of an architecture. Software architectures provide a great deal of flexibility in combining the different parts together. Consequently, it is very important to provide proper assembly instructions for software architectures.

STUDENT INTERACTION

What are the possible types of homes we can build using the given architecture?

OBJECTIVE

Students should be able to discuss the importance of assembly instructions when building new applications using the architecture.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Examples of Instructions

Sound System
Instructions

Entertainment
Center
Instructions

7-9

DISCUSSION - Software Architecture Documentation

As you probably know, software parts and software systems can be very complex. Consequently, instructions for assembling a set of software parts are crucial to building a software system.

Just as some mechanical parts cannot operate with other parts (a DC battery cannot turn an AC motor without the help of a transformer), some software parts do not work with certain other software parts.

Software assembly instructions should state clearly the possible interactions and any limitations (also known as constraints) among different software parts. Where possible, workarounds for identified problems should either be supplied or suggested.

This slide depicts an example of a software architecture and an outline of the descriptive documentation for that architecture.

STUDENT INTERACTION

When you write software, the code should be all the documentation that you need. Discuss.

OBJECTIVES

Students should be able to:

- Clearly understand that a set of instructions or documentation is a necessary part of software development
- Give three reasons why software requires instructions

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Software Architecture Documentation
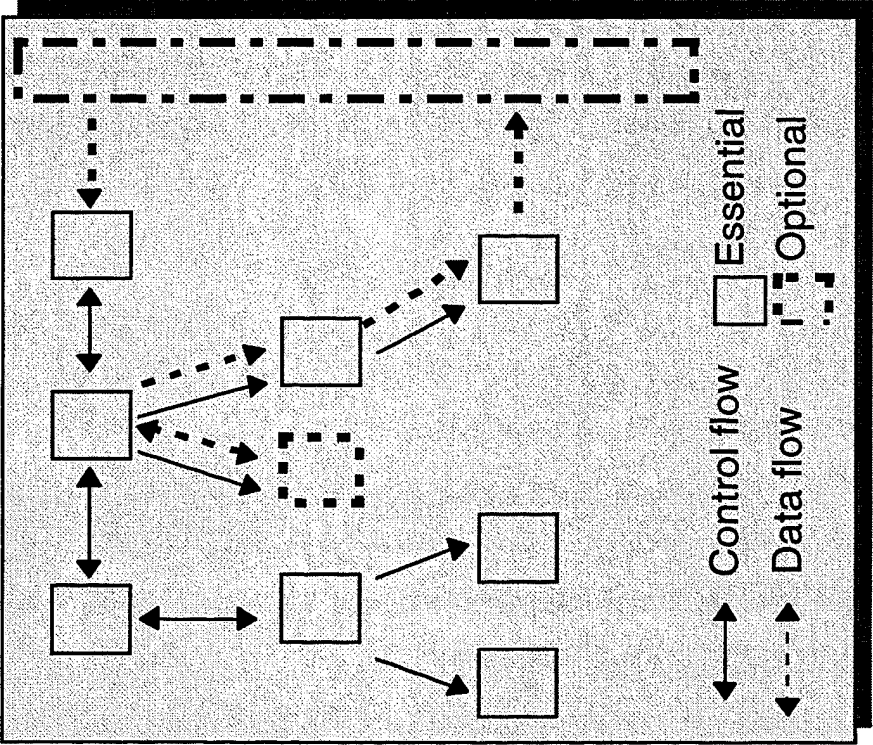
1.0 General Constraints

2.0 Component Descriptions

2.1 Component A

2.1.1 Connections

2.1.2 Constraints

2.2 Component B

3.0 Lessons Learned

Control flow — Essential

Data flow --- Optional

SOFTWARE PRODUCTIVITY CONSORTIUM

DISCUSSION - Factors to Consider

So far, we have gained an understanding of what an architecture is and what its components are: style, parts, and assembly instructions. Now, let's think about how these architectures come into existence.

Architectures are not defined overnight. They are continually modified, and they evolve over time. As people learn new things (sometimes by correcting mistakes), the architecture is refined. We can see that happening in cars even today. Airbags and anti-locking brakes are recent modifications. Architecture embodies the knowledge that is accrued over a long period of time by a large number of different people. It would be a waste of time to throw all this knowledge away.

Let's look at several factors that must be considered when we either define a new architecture or refine an existing architecture. Generally, these factors can be grouped under one of the following: business/monetary factors, technical or engineering factors, and social factors. Let us explore the impact of each of these factors when defining our robot architecture:

- Business factors: Defining and supporting a domain architecture requires up-front investment to improve significantly the chances for future cost savings while developing new applications. Suppose it would cost $15,000 to build a robot to pick corn and another $15,000 to build one to pick only tomatoes. As the manager of robots, you estimate that it will cost $100,000 to define and support a common architecture for robots that can pick any kind of vegetable. How many robots do you need to sell in order to recoup your initial investment? Discuss the three business factors on the slide in terms of robot manufacturing approaches.
- Technical factors: Suppose, for stability reasons, a robot cannot be taller than 5 feet. It would not be able to reach certain fruits or vegetables that grow too high.
- Social and other factors: Your architecture should adhere to certain norms that are either explicitly (or implicitly) imposed by society. For example, certain states may decide that all robots should use solar power. That would change the architecture of our robot, would it not? In some countries, people drive on the left side of the road, and, because of this, the steering wheel is placed on the right side of a car.

In summary, many factors affect how you define an architecture, and you must take into consideration all the factors while designing an architecture.

STUDENT INTERACTIONS

- Why should you build and support an architecture?
- Is it technically feasible to design a robot architecture to pick any vegetable?
- What other social factors might we consider in defining our robot's architecture?

OBJECTIVE

Students should be able to list three factors to consider when defining an architecture.

7-11

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Factors to Consider

Social Factors

Car With Steering Wheel on Left

Car With Steering Wheel on Right

Business Factors

- Cost of Building
- Cost of Maintenance
- Number of Customers

Coconut Tree

Can't get there from here!!

Technical Factors

SOFTWARE PRODUCTIVITY CONSORTIUM
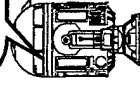
DISCUSSION - Unit Summary

In the beginning of this unit, we stated that the most important product of domain design was software architectures. We have discussed architectures and software architectures in particular.

Domain designers take the information gathered and organized during domain analysis, and they create a formal solution. Domain design consists of designing a domain architecture and developing guidelines for the use of this architecture.

An architecture is a well-established way of building solutions to certain problems. Architectures evolve over a period of time. This evolution is necessary to keep pace with changes in technology, as well as changes in the needs of the customers.

At a high level, every architecture has three attributes or components: style, parts, and assembly instructions.

- Different architecture styles exist to solve different kinds of problems within the same problem area. Styles are just variations of the same basic architecture.
- Each architecture is a collection of parts. Detailed descriptions should be given of each and every part of the architecture.
- Assembly instructions must be prepared for the chosen architecture style. The instructions address the various ways of putting together the parts of an architecture.

All the attributes and general ideas about architectures apply fully to software architectures. The software industry is defining software architecture styles that suit specific needs. Since the software industry is a young industry, its architecture styles are still evolving. A software architecture defines the software parts and how they work together. The software assembly instructions should clearly state the possible interactions among and any limitations of different software parts.

Several factors must be considered when we either define a new architecture or refine an existing architecture. Generally, these factors can be grouped under one of the following: business or monetary factors, technical or engineering factors, and social factors.

OBJECTIVES FOR THE UNIT

Students should be able to explain the role of architectures in domain design.

7-12

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Unit Summary

- **Architectures are very important to domain design.**

- **Architectures evolve.**

- **Architectures have three attributes:**

  — **Style**

  — **Parts**

  — **Instructions**

- **Software architectures have the same attributes.**

- **Many factors influence and constrain architecture development.**

SOFTWARE
PRODUCTIVITY
CONSORTIUM

UNIT 7: SOFTWARE ARCHITECTURES

Summary

Domain designers take the information gathered and organized during domain analysis, and create a formal solution. As with domain engineering in general, domain design is not a clear-cut, step-by-step process. It is a set of iterative, high-level activities. In general, domain design consists of designing a domain architecture and developing guidelines for the use of this architecture.

An architecture is nothing but a well-established way of building solutions to certain problems.

Architectures evolve over a long period of time. They reflect the wisdom of many experts. Errors in the architecture have been corrected over time by a large number of people. The strengths and weaknesses of architectures are well known by their user community. They can save large amounts of resources, time, money, and people when used over and over again.

By definition, an architecture satisfies the needs of a number of different people. Are all people's needs the same? No, they may be similar, but they are not the same in all cases. Usually there are minor differences. Consequently, an architecture must accommodate these small variations among different user needs. If the variations among user needs or groups of user needs are too varied, different architectures may be necessary.

At a high level, every architecture has three attributes or components: style, parts, and assembly instructions.

Different architecture styles exist to solve different kinds of problems within the same problem area. Styles are just variations of the same basic architecture. Even though you may see different architecture styles, they all share many commonalties with each other.

Each architecture is a collection of parts. For each style of architecture, the different parts that make up that architecture can be seen clearly. Since it is our job to help others build their own systems, we must provide a complete description of each part of the architecture.

Detailed descriptions should be given for each and every part of the architecture. These descriptions help not only in building the parts from scratch (to write brand new software code, in the case of software architectures), but also in evaluating existing pieces to see if they can be used with the current architecture.

We now understand that an architecture is made up of several different parts. It is very important to note that these parts are connected to each other as defined by the architecture. The interconnections between the parts are what is of interest to our customers.

If we are going to help others build something, it is not enough just to provide them with an architecture style and a list of parts. In a way, assembly instructions provide the "glue" that is necessary to put all the parts together.

Assembly instructions must address the different ways of putting together the parts of an architecture. Software architectures provide a great deal of flexibility in combining the different parts. Consequently, it is very important to provide proper assembly instructions for software architectures.

Architectures are not defined overnight. They are continually modified and evolve over time. As people learn new things (sometimes by correcting mistakes), the architecture is refined. It would be a waste of time to throw all this knowledge away.

Several factors must be considered when we either define a new architecture or refine an existing architecture. Generally these factors can be grouped under one of the following: business/monetary factors, technical/engineering factors, and social factors.

All of the general ideas and attributes about architectures apply fully to software architectures.

Exercise

Review

1.   List the three components of an architecture.

2.   List the three factors that impact an architecture.

Define a Software Architecture for the Robot

3.   What is the first step in solving a customer's problem?

Examples of customer's problems using a robot (a) to pick vegetables, (b) to search for missing persons, and (c) to participate in planetary explorations.

4.   What do we use to describe these problems?

Identify Customer Needs

5.   Suppose a farmer needs a robot to work throughout the night to meet his delivery schedules, but his state has recently passed a law requiring use of solar power for all outdoor robots. Could we support this request? Why or why not? Would our decision support the needs of our other customers? Explain. Is there a workaround to this problem?

At the end of this step, let's assume we have decided that our robot will support the following needs:

- Harvest between 50 and 500 ears of corn

- Begin its mission at a specified origin and return to that origin on mission completion

To support the above needs, our robot should be able to do the following things:

- Move from place to place

- Pluck and pick things up (with arms with fingers)

- Identify different kinds of objects

- Know the shortest and safest distance between two points

- Communicate with humans, other robots, or other communication mechanisms

- Power pack (preferably renewable)

- Optional: a bin to carry back vegetables from the farm

We can also make this a core of this multipurpose robot. The bin can be used to carry emergency supplies (food, warm clothes, etc.) for the lost people, and it can be used to bring rock samples back to the spaceship while conducting planetary explorations.

Select an Architecture

6. In the table below, list for each need the advantages and disadvantages of selecting each of the three architectures.

Needs	R2D2		Humanoid		Spider	
	pros	cons	pros	cons	pros	cons

Select the Parts

7. What information do you think the robot "brain" needs to do its job?

8. Is the "brain" a physical part, like a leg or an arm?

9. List all of the parts needed by the robot.

Group Exercise

Hardware group attempts to answer the following questions:

10. What kind of materials should be used for the robot's arms and legs?

11. What kind of cables should be used?

12. What types of batteries should be used?

13. Where should the computer be installed?

Hardware group interacts with the software group to answer the following questions:

14. What kind of computer is needed to run the software?

15. How much memory is needed on the computer to run the software?

16. What kind of "eyes" are needed to give the right kind of information to the robot "brain"?

Record other questions (and their answers) as you discover them.

Software group attempts to answer the following questions:

17. What does our software need to do?

18. What parts can be logically partitioned?

19. What parts are currently available?

20. What hardware/computer does it need to run on?

21. How much memory do we have available?

Write Software Code

22. For each of the architectural components shown in Figure A, list the functions that should be provided.
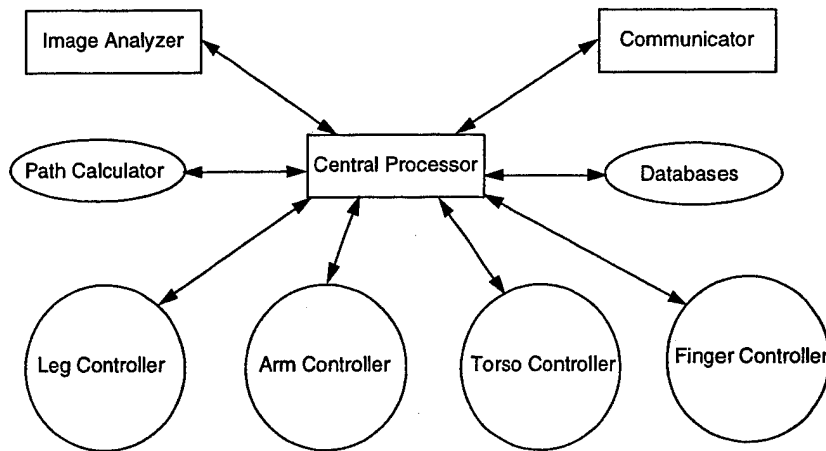
Figure A

Test

Once we think our robot has been sufficiently tested in the laboratory, we should take it to a farm and see how it performs. We may find that our robot is holding the tomatoes too tight and thus crushing them in the process.

23. Is this a hardware or software problem?

24. How about testing the robot to see if it will perform well in planetary explorations?

25. How do we test this? (simulation of conditions)

26. List the steps involved in defining an architecture for the robot.

Teacher's Notes for Exercises

Exercise

Review

1. List the three components of an architecture.

Style, parts, and instructions

2. List the three factors that impact an architecture.

Business, technical, and social

All the knowledge we have accumulated so far about architectures in general also applies to software architectures.

Define a Software Architecture for the Robot

Notes: In this practical exercise, students define a software architecture for the robot based on the domain model developed during the domain analysis segment of this course. A suggested script follows.

3. What is the first step in solving a customer's problem?

As you have learned, the first step in solving a customer's problem is to <u>state the problem precisely</u>. But we are trying to solve several customers' problems (to pick vegetables, to search for missing persons, and to participate in planetary explorations).

4. What do we use to describe these problems?

The domain model is the precise statement of the problem we are trying to solve.

Identify Customer Needs

Now, let's look at the different needs of the customers as defined in the domain model. We have to decide which needs we will satisfy using our robot. This is where we will apply business and technical factors to see which needs we can satisfy and which ones we cannot. For example, one of the farmers has a coconut grove and wants help in plucking the coconuts off the tall trees. We may decide that we cannot support this need, because we would have to build a very tall robot to be able to see and pluck the coconuts. Additionally, we may not be able to use this tall robot to pick tomatoes (the tall robot may not be able to see the tomato plants well and may trample them).

To get around this problem, we can suggest to the coconut farmer that he employ someone to climb the trees and pluck the coconuts from the trees. The employee can then leave the coconuts in a bin at the bottom of the tree. Then our robot can pick up

the coconuts from the bins and carry them back to the storage facility. You can see that a given problem is usually solved by a combination of manual and automated solutions.

In this fashion, we perform a detailed analysis of all the needs in the domain model and decide on which needs we can support using the architecture and which needs we cannot or will not.

5.   *Suppose a farmer needs a robot to work throughout the night to meet his delivery schedules, but his state has recently passed a law requiring use of solar power for all outdoor robots. Could we support this request? Why or why not? Would our decision support the needs of our other customers? Explain. Is there a workaround to this problem?*

At the end of this step, we have decided that our robot will support the following needs:

- Harvest between 50 and 500 ears of corn

- Begin its mission at a specified origin and return to that origin on mission completion

To support the above needs, our robot should be able to do the following things

- *Move from place to place*

- *Pluck and pick things up (with arms with fingers)*

- *Identify different kinds of objects*

- *Know the shortest and safest distance between two points*

- *Communicate with humans, other robots, or other communication mechanisms*

- *Power-pack (preferably renewable)*

- *Optional: a bin to carry back vegetables from the farm.*

We can also make this a core element of this multipurpose robot. The bin can be used to carry emergency supplies (food, warm clothes, etc.) for the lost people, and it can be used to bring rock samples back to the spaceship while conducting planetary explorations.

Select an Architecture

6. *In the table, list the advantages and disadvantages of selecting each of the three architectures.*

Needs	R2D2		Humanoid		Spider	
	pros	cons	pros	cons	pros	cons
	small	unstable	average stability		very good stability	may be large & clumsy for farmers' needs
	compact	can't carry heavy items				leg control is complex
	cheaper to build?					
	looks cute				frighens crows	frightens cows

The next step is to decide which robot architecture is best suited to support the needs we have selected. Here again we have to apply the factors we talked about earlier to help us make our decisions. For example, clearly we cannot use the egg-shaped robot architecture. Because of its instability, it cannot effectively support the needs of planetary explorations and travel through the Arctic tundra.

The decision should lead to either a humanoid or a spider robot. If we choose a spider, we must make sure it is not too large for use by the farmers.

Select the Parts

Now we are ready for the next step. We have decided on the ___R2D2___ architecture, and now we have to make sure that our robot can carry out all the tasks that we listed in __Step 5.__ We also have to assign tasks to the different parts of the architecture. Let us start analyzing the tasks and see which parts of the robot can take care of which tasks.

>*Move from place to place. The robot has to be able to traverse difficult terrain.*
>
>As we analyze this task, we find out that our robot can move from one place to another using its legs. But who will control and decide where to put the next step? We know that our brains take care of the problem for us, so we need some kind of "brain" for our robot.

7. *What information do you think the robot "brain" needs to do its job?*

The brain needs images of the environment to decide what to do next. Just as our eyes provide this information to our brains, we need to connect the eyes of the robot to its brain. Now we have decided on a new part for our robot, namely its brain.

8. *Is the "brain" a physical part, like a leg or an arm?*

Actually, it is all software code (lots of it). Additionally, we need one or more computers (more new parts) to support this new robot brain.

>Let us pause and take a look at what we have done so far. As we started analyzing the different tasks that have to be carried out by our robot, we have also started identifying the different parts that are needed to carry out the tasks. We have also seen that some parts of our robot architecture can be purely physical (such as legs and arms) and some parts can be purely software parts ("brain"). In general, our robot architecture consists of physical parts (also called hardware parts) and software parts.

9. *List the parts needed by the robot.*

Physical parts:

*   Head, arm, foot, torso, brain, bin

Software parts:

*   Image analyzer, communication, central processor, leg controller, arm controller, torso controller, finger controller

Group Exercise

Once we have decided on all the parts, both software and hardware, we can split the work into two major interacting tasks. One group can work on defining the hardware/physical architecture and the other group can work on defining the software architecture. This is how it is normally done in the software industry. The two groups have to work very closely so that, in the end, both software and hardware parts will work with each other, just as our brain (software) works with our different body parts (hardware).

The hardware group will be concerned with mechanical and engineering aspects of building the robot: what kind of material to use for robots arms and legs, what kind of cables to use, what types of batteries to use, where to install the computer, etc. Of course, this group has to interact with the software group to make sure that the right kind of computer is used for the software, to be sure enough memory is available is needed on the computer to run the software, to determin what kind of "eyes" are needed to give the right kind of information to the robot "brain" and so on.

Hardware group attempts to answer the following questions:

10.   What kind of materials should be used for the robot's arms and legs?

11.   What kind of cables should be used?

12.   What types of batteries should be used?

13.   Where should the computer be installed?

Interact with the software group, by answering the following questions:

14.   What kind of computer is needed to run the software?

15.   How much memory is needed on the computer to run the software?

16.   What kind of "eyes" are needed to give the right kind of information to the robot "brain"?

Record other questions (and their answers) as you discover them.

The software group will be concerned with designing a software architecture for the brains of our robot. Just as the hardware group defines different parts in the hardware architecture (arms, legs, eyes, etc.), the software group defines the different parts in the software architecture. These parts have to map to the physical parts for our robot to work. For example, we can define an image analyzer that gets information from the eyes of the robot and determines what objects the robot is currently looking at. We can also define a software part called leg controller that controls the movement of robot legs.

Software group attempts to answer the following questions:

17. What does our software need to do?

18. What parts can be logically partitioned?

19. What parts are currently available?

20. What hardware/computer does it need to run on?

21. How much memory do we have available?

Figure A shows a preliminary version of our software architecture.
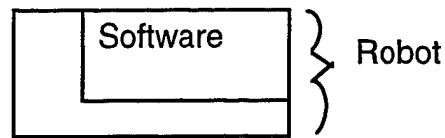
Figure A

When defining an architecture, we have to keep some important things in mind:

- It has to be simple.

- It should make use of existing parts.

- Each part should have a specific role in the architecture.

For example, we can draw a single box that does everything that the software has to do. This is not of much use. This box will not simplify the building of a new application.

Let's look at the second picture. Which one of these two pictures gives you a better idea of the kind of software needed for our robot?
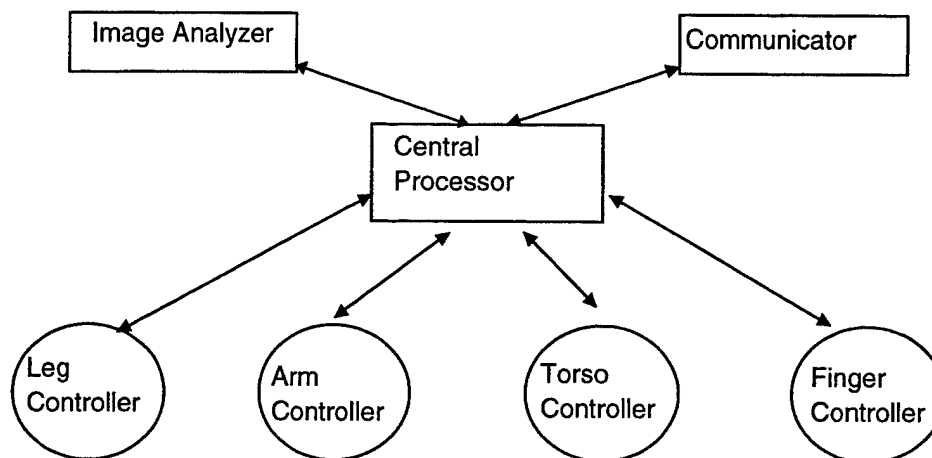
Figure B

What do you think each of these parts do?

The central processor has a lot to do. For each of the images the image analyzer identifies, the CP has to decide what to do next: go forward if vegetable, human, etc.; avoid it (boulder, streams, etc.); ignore it (distant objects, leaves, etc.). If it sees an object of interest (such as a human or vegetable), the CP has to compute the safest and shortest way to get there. Also, the CP needs a memory to do the job (it has to "know" what an onion looks like, what a boulder looks like, etc.)

So, we may decide to break up the CP into smaller parts, such as path calculator, memory/database, and central processor (a smaller version of our original one). Now, our architecture will look like this:
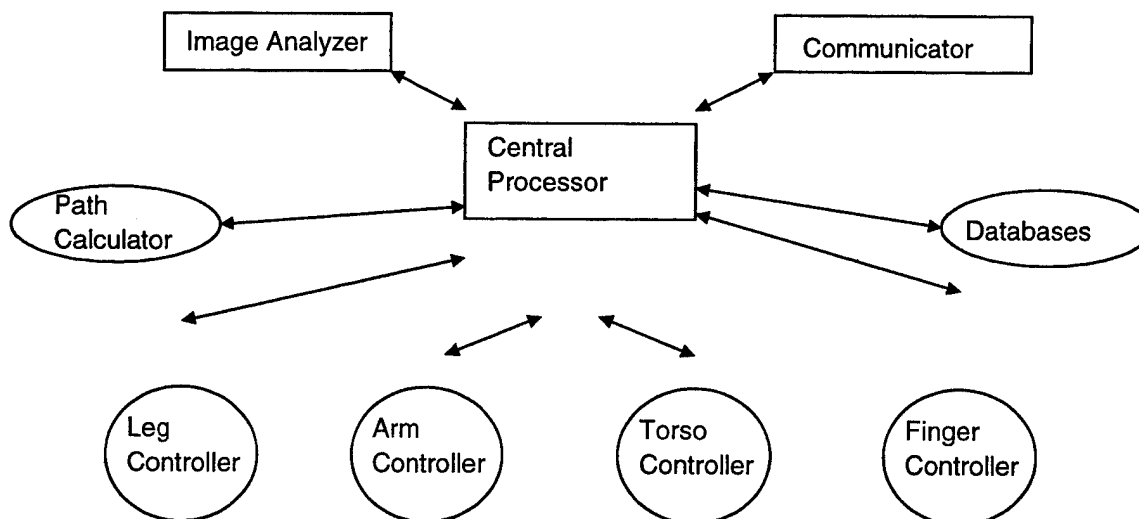
Figure C

We have to keep breaking down each part until we are satisfied that the parts are small enough or we can find existing parts.

Write Software Code

Once all the software parts have been identified and clearly defined, we have to write the software code for each part of the software architecture. Why do we have to do this? We have to make sure that our robot will work the way we intended it to work.

After writing the software, we can load our software onto the robot's computer and test if the software works correctly with the physical parts of the robot. We may find a lot of mistakes in our software (called "bugs" in the software industry) as well as hardware. Both hardware and software groups will have to work with each other to fix all the problems.

Test

Once we think our robot has been sufficiently tested in the laboratory, we may decide to take it to a farm and see how it performs. We may find that our robot is holding the tomatoes too tight and crushing them in the process.

23. Is this a hardware or software problem?

It could be either or both. We may also find that the robot is picking tomatoes before they are ripe. (Where does the problem lie? Hint: the software controlling the image analyzer.) We have to test our robot in the forest, too, and see how it performs.

24. How about testing the robot to see if it will perform well in planetary explorations?

25. How do we test this? (simulation of conditions)

We have to make sure that our robot architecture, both hardware and software, has been sufficiently tested before we can provide the robots to our customers. Once we offer them to our customers, periodically we have to monitor our robots and fix any problems. We may also upgrade the software as well as hardware, based on the comments we get from our customers.

26. List the steps involved in defining an architecture for the robot.

Identify customer needs

Select an architecture

Select the parts

Write software code

Test

DISCUSSION - Unit 8: Writing Reusable Software

In this unit, we will discuss some considerations and issues that are involved when writing software code for reuse, that is, writing software that satisfies the needs of more than one customer.

So far, we have seen how to build and use an architecture. When building any application, we need to know the style and have the parts and assembly instructions. In this unit, we will talk about the parts. As you already know, in the software industry, parts are nothing more than software code.

OBJECTIVES FOR THE UNIT

Students should be able to:

- Understand the need to write reusable software
- Know some of the characteristics of reusable software

8-1

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Unit 8: Writing Reusable Software

SOFTWARE
PRODUCTIVITY
CONSORTIUM

8-1

DISCUSSION - Reusable Architecture

Without software code, we cannot build the applications we want.

You may wonder why we are talking about software code again—we already know how to write software code. But there is an important distinction. Historically, when software code was written, it was written to satisfy a particular need of a customer. Now the trend in the software industry is to write software code that satisfies the needs of more than one customer.

The example on the slide shows that different calculators perform different functions. If the code is written in a generic fashion, or in modules, then more than one customer can use one or more of the modules.

STUDENT INTERACTIONS

- Is there a difference between software code written for architectures versus software code that we normally write? Why?
- What examples can you think of where something is done or built to satisfy particular needs of the customer?
- What do you think are the advantages of writing software code that satisfies the needs of more than once customer?

OBJECTIVES

Students should be able to:

- Recognize that the new trend in the software industry is to write reusable software
- State some obvious benefits from writing reusable software

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Reusable Architecture

Generic Architecture: Calculator

Other Calculator Functions:

- Turn Power On
- Turn Power Off
- Store Data Memory
- Recall Data Memory
- Print Data
- Display Data

Advanced Math Functions:

- Trigonometric
- Statistical

General Math Functions:

- Add Data
- Subtract Data
- Multiply Data
- Divide Data
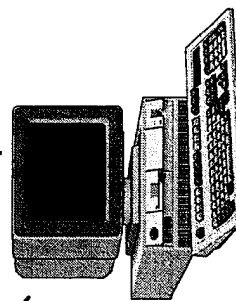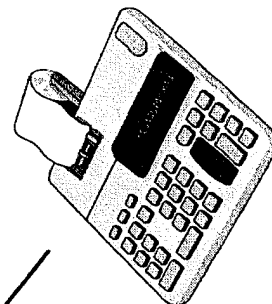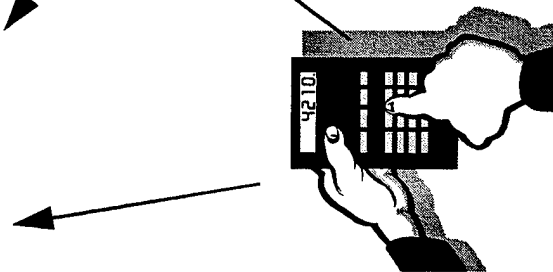
DISCUSSION - Options in Reusing Software

From our previous discussions, we know that an architecture solves the problems encountered by a number of people. The architecture is made up of different parts. Each part has a specific function to fulfill. For example, the image analyzer part in our robot architecture must do certain things. If we write software code that analyzes images, this software can be used repeatedly by other people building robots—and possibly working in other domains.

First let's look at the options available to us in writing software code that analyzes images:

- We can look at software code already available (from previous projects or commercial software) and see if the software code can be used directly as an image analyzer. Sometimes we may have to combine two or more software pieces to get what we want.

- Sometimes we may find software code that does most of what we want, but lacks some aspect. For example, we may find an image analyzer that cannot identify colors (so it could not tell a ripe tomato from a green one). In such a case, we can enhance the image analyzer by writing additional software and then using it. Sometimes, though, it may not be feasible to change the existing software.

- In other cases, we may not find any existing software, and we have to write brand new software. Once written, this software code can be used by other people building robots. Additionally, this software code may be used by people in other domains that have a need for image analysis.

We repeat this process for every part of our robot architecture.

STUDENT INTERACTIONS

- For what other applications can you picture the use of the image analyzer besides robots?
- What are some reasons we may not be able to modify existing software?

OBJECTIVE

Students should be able to define the various alternatives of developing software to support an architecture.

8-3

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Options in Reusing Software

- Use the entire piece as is.

- Take the useful pieces and put them together.

- Take the useful pieces and add new code.

- Write entirely new code.

8-3

DISCUSSION - Writing Reusable Software

When you write reuseable software, you must make some effort beyond that required for software you do not expect to reuse:

- Each software part fits into an architecture. You must make sure it will fit into any architecture in which it might be reused.
- Each part must be clearly identified so that its purpose is apparent. This will increase its chances of being reused.
- Each part must be carefully developed so as to ensure that it does not make any assumptions about how it is used that might limits its general utility. For example, people who write procedures that access global variables limit the use of those procedures, because any program that uses them must also have those same global variables.
- Each software part must be tested not just with test cases based on how it is used in a particular program, but on how it might be used in any program. For example, one program that uses a square root function might be careful never to call that function with a negative argument. Other programs might not be so careful, so the square root function should be tested with both negative and positive arguments.

STUDENT INTERACTION

What should you do before you decide to write software that you think will help lots of people?

OBJECTIVES

Students should be able to:
- Understand that developing reusable software requires forethought
- List some properties that reusable software possesses

8-4

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Writing Reusable Software

Reusable Software Components:

- Must be carefully designed based on overall architecture

- Must be carefully identified

- Must be carefully developed

- Must be thoroughly tested to cover all use situations

8-4

SOFTWARE
PRODUCTIVITY
CONSORTIUM

DISCUSSION - Megaprogramming

Now let's tie the material we've studied to how application engineers develop software using megaprogramming. Software development starts with someone (a customer) recognizing a problem and asking an application engineer to solve it. The application engineer uses the knowledge gained during domain analysis to create a prescriptive model of the problem. This prescriptive model serves as a precisely-defined problem.

The application engineer uses this precisely-defined problem to produce a solution to the customer's problem. The application engineer selects an appropriate architecture style from the knowledge of past solutions. He or she then selects parts that fit this architecture style, and follows the instructions to assemble those parts.

STUDENT INTERACTION

How does this differ from how you develop software?

OBJECTIVES

Students should be able to:
- Understand how domain engineering supports application engineering

8-5

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Megaprogramming

Application Engineering

Vaguely Defined Problem → Application Engineer → Precisely Defined Problem → Application Engineer → New Solution

Domain Engineering

Knowledge of Past Similar Problems — Domain Analysis

Knowledge of Past Solutions — Domain Design

SOFTWARE PRODUCTIVITY CONSORTIUM
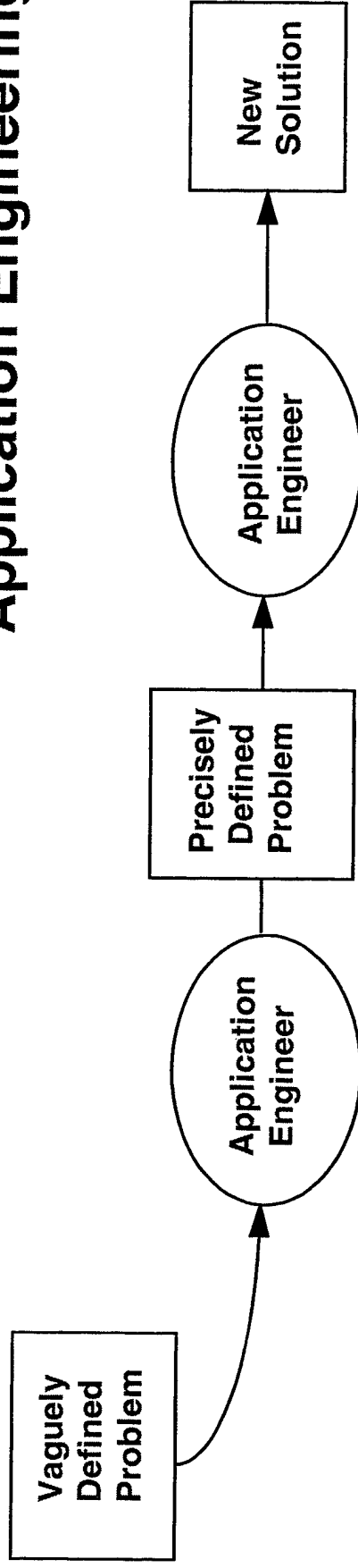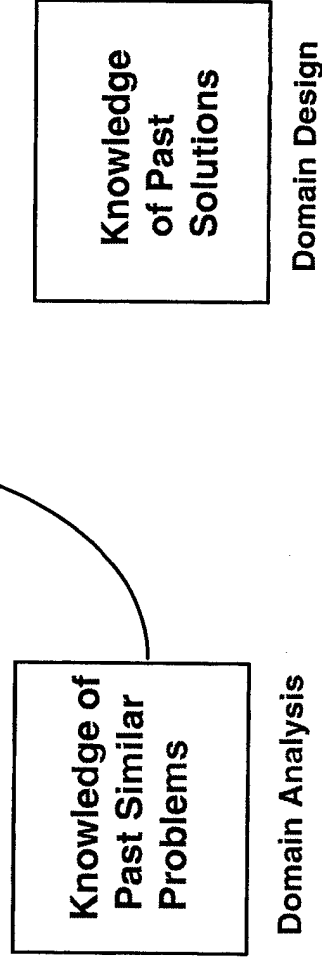
DISCUSSION -Unit Summary
Let's summarize why writing software code is different today from what it has been in the past.

At one time, software was written to meet a particular customer's needs without considering that someone else might need similar, but not identical, software. In other words, software was often hard-coded to the specific needs and constraints of the customer.

Now software code is written to satisfy multiple users to improve productivity and cut costs. As the benefits of software reuse have become more apparent, the attempt to generalize code in order to reuse it on future projects within an organization has become more common.

However, there are important distinctions between writing software for single users versus multiple users. New procedures must be developed to meet these new needs.

Domain engineering is an extension of the concept that allows software to be reused across organizations within a specific area of interest. The tendency in the software engineering industry in recent years has been to make software easily understandable by both the users of the software and those who will maintain the software in the future.

OBJECTIVES

The student should be able to:

- Describe why reusable software code is important

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Unit Summary

- **Domain design: writing reusable software:**

  — **Write software that satisfies the needs of many people.**

  — **Use existing software, if it is useful.**

  — **Design the software to be reusable.**

  — **Write useful, reusable software and spread joy!**

UNIT 8: WRITING REUSABLE SOFTWARE

Summary

We have seen how to build and use an architecture. When we build an application, we need to know the style and have the parts and assembly instructions. In this unit, we talk about the parts. In the software industry, the parts are nothing more than software code.

Without software code, we cannot build the applications we want.

Historically, when software code was written, it was written to satisfy a particular need of a customer. Now the trend in the software industry is to write software code that satisfies the needs of more than one customer.

We know that an architecture solves the problems that are encountered by a number of people. The architecture is made up of different parts. Each part has a specific function to fulfill. If we write software code that analyzes images, this software can be used repeatedly by other people building robots—and possibly working in other domains.

When you write software code to support an architecture (or support multiple customers), you must take into consideration several needs. You might think that this can be costly and cumbersome. You are right! But once you write this special software, you have to write new software every time another new customer comes to you. You will save time and money in the long run.

One key thing to remember is that almost all software code that was ever written and used **changes**. This is usually because the needs of the customer change or evolve. A large proportion of the costs involved in the software industry is attributable to changes.

It is very important to write software code that allows for changes to be made easily. We must write code in such a manner that people other than just the programmer can understand it easily. When writing software code, we must be sure to provide at least the following:

- Detailed "description for people who use" the software.

- Detailed "description for people who may change" the software, including yourself!

Writing software code is different today from what it was in the past. In the past, software was written to meet a particular customer's needs without considering that someone else might need similar, but not identical, software. In other words, software was often hard-coded to the specific needs and constraints of the customer.

Now software code is written to satisfy multiple users to improve productivity and cut costs. As the benefits of software reuse have become more apparent, the attempt to generalize code in order to reuse it on future projects within an organization has become more common.

However, there are important distinctions between writing software for single users versus multiple users. New procedures must be developed to meet these new needs.

Domain engineering is an extension of the concept that allows software to be reused across organizations within a specific area of interest. The tendency in the software engineering industry in recent years has been to make software easily understandable by both the users of the software and those who will maintain the software in the future.

Exercises

The Difficulty of Writing Reusable Software

1. Write the pseudocode necessary to determine the "keyword" in the following line of code:

    Print "Hello"

    (Print is the keyword; "Hello" is a parameter.)

2. Write additional pseudocode to parse out "keywords" and "parameters."

3. Write one piece of pseudocode that will do both.

Teacher's Notes for Exercises

The following exercises are designed to show students that it is more difficult to design for reuse rather than to design software just for a single use.

The Difficulty of Writing Reusable Software

1. *Write the pseudocode necessary to determine the "keyword" in the following line of code:*

   Print "Hello" (print is the keyword; "Hello" is a parameter.)

   Get letter until a space is encountered

   Concatenate each letter

   Match string to set of "keywords"

2. *Write additional pseudocode to parse out "keywords" and "parameters."*

   Get letter until a space is encountered

   Concatenate each letter to form a "keyword" string

   Match string to set of predefined "keywords"

   For each "keyword"

   Get letter until an appropriate delimiter is encountered

   Concatenate each letter to form a "parameter" string

3. *Write one piece of pseudocode that will do both.*

   Get character until a space is encountered

   Concatenate each letter to form an "unknown" string

   Match "unknown" string to set of predefined "keywords"

   If no match is found, "unknown" string is a parameter

DISCUSSION - Unit 9: Summary of Domain Design

In this phase of the course, we have discussed the exciting field of domain design and software architectures, which represent the cutting edge of software technology.

We covered the following topics:

- Megaprogramming and domain analysis, and their relationship to domain design

- Differences between problems and solutions. Domain design products provide solutions to problems first defined using domain analysis.

- Architecture, the most important product of domain design, especially software architecture.

- New ways of writing software code to support megaprogramming principles, especially software reuse.

The field of domain design is new and will rapidly develop. Its principles and concepts are of increasing importance in this age of information.

OBJECTIVES FOR THE UNIT

Students should be able to:

- Discuss the role of domain design in megaprogramming
- Describe the functions associated with domain design

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Unit 9: Summary of Domain Design

SOFTWARE
PRODUCTIVITY
CONSORTIUM

DISCUSSION - Summary of Domain Engineering

Domain engineering is a process that analyzes the existing software environment in an organization in order to derive products that application engineering can use to develop current software solutions. Domain engineering consists of two major subprocesses or phases:

- Domain analysis involves scoping a domain, identifying various systems within the domain; modeling the requirements and functions for individual systems within the domain; combining individual system models into one comprehensive descriptive model; and developing a prescriptive model, which documents future system requirements and functions.
- Domain design involves developing a generic design for solutions to problems in the domain.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Summary of Domain Engineering

- Domain engineering is composed of:

  — Domain analysis

    – Domain scoping

    – Individual system models

    – Descriptive model

    – Prescriptive model

  — Domain design

    – Problems and solutions

    – Architectures

    – Reusable software

DISCUSSION - Summary of Domain Analysis

Domain analysis defines the problem in terms of the requirements and supporting functions of a domain in an easily representable, user-friendly, and flexible manner. The results of domain analysis should allow organizations to adapt the domain models in terms of future technology, time, requirements, personnel, and budget.

9-3

Summary of Domain Analysis

- Domain analysis allows you to:

  — Collect functions and requirements on different systems in a domain

  — Organize those functions and requirements into models that represent past, present, and future systems

9-3

DISCUSSION - Summary of Domain Scoping

Domain scoping is merely the application of an activity you have applied in the past—identifying a small, manageable problem within a large, complex problem.

We have discussed how domain scoping can be accomplished. First, the set of possible domains (the domains of interest) are modeled. Next, commonalities are found between the models. One way to do this is to look for common functions. Next, the small, manageable problem—the domain of focus—must be selected. This can be done either by a management decision or based on selection criteria if there is more than one common area.

Another model, the context diagram, is then produced to describe the domain of focus graphically. A textual description should accompany this graphic.

Finally, the results of this process, especially the context diagram, are verified with the customer.

9-4

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Summary of Domain Scoping

- Domain scoping is needed to:

  — Reduce a large unmanageable set of potential needs, options, and alternatives to a manageable size

  — Define the domain and potential products precisely

- Functional models for DOIs can be derived from requirements.

- The DOF can be one of the common functions.

- A context diagram refers to the DOF precisely.

- Verification increases the success rate.

DISCUSSION - Summary of Individual System Models

Problems solved in the past offer information that can still be important and useful today. The entire past problem and its solution may not be exactly the same as the current problem. However, when aspects of the past problem could be similar to aspects of the current problem, aspects of the past problem's solution are likely to support the solution of the current problem.

An individual system model describes an individual system from a past problem and represents it as a graphical model. The key is to model as many systems as necessary to describe effectively the common and different requirements and functions within a domain.

9-5

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Summary of
Individual System Models

- Represent the common and different functions and requirements of systems within a domain

- Represent the current system, not new systems

9-5

SOFTWARE
PRODUCTIVITY
CONSORTIUM

DISCUSSION - Summary of Description Model

A combined, or descriptive, model is a graphical representation of the common and different functions, features, and requirements for all individual system models. This is the key model that domain design uses to create generic reusable software components. During domain design, the common functions are organized further into software modules that represent routines that a software system must perform. The goal of domain analysts is to create descriptive models for all the systems in an entire domain so all software functions can be identified.

9-6

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Summary of Descriptive Model

- Combines the common and different requirements and functions from individual system models

- Represents information from individual system models as a single model

- Common aspects suggest areas of possible reuse

SOFTWARE
PRODUCTIVITY
CONSORTIUM

DISCUSSION - Summary of Prescriptive Model

The prescriptive model, also known as the to-be model, is simply an extension of the descriptive model. The descriptive model is based on old problems and their unique solutions. When a new system is to be designed, certain of its functions overlap with old functions; to show these relationships, create the descriptive model. However, the new system will require more updated functions, if not entirely new ones. The prescriptive model allows the engineer to estimate which updates or enhancements may be needed to accommodate future systems requirements.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Summary of Prescriptive Model

- Enhances descriptive model with requirements for future systems

- Hand-off model to domain design

9-7

SOFTWARE
PRODUCTIVITY
CONSORTIUM

DISCUSSION - Domain Design: Problems and Solutions

In this course, we have begun to understand the exciting field of domain design and software architectures, which reflect the cutting-edge in software technology. Let us review what we have learned so far in this course. Domain design uses the information generated during domain analysis to define a solution to problems in a domain. This is done by developing a domain architecture and usage guidelines for the architecture.

Domain Design:
Problems and Solutions

- Solution to a set of problems

- Uses prescriptive model generated by domain analysis effort

- Must reflect solutions to requirements modeled during domain analysis

SOFTWARE
PRODUCTIVITY
CONSORTIUM

DISCUSSION - Domain Design: Architectures

An architecture has three components: a style, a set of parts, and instructions. The style defines the parts and how they can interconnect. The instructions tell how to accomplish that interconnection.

Just as building architectures evolve, so do software architectures, and for the same reasons. They evolve in response to improved understanding of the parts and how they are interconnected. They also evolve in response to external factors, i.e., how they are used.

9-9

Domain Design: Architectures

- Architectures are very important to domain design.

- Architectures evolve.

- Architectures have three attributes:

  — Style

  — Parts

  — Instructions

- Software architectures have the same attributes.

- Many factors influence and constrain architecture development.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

DISCUSSION - Domain Design: Writing Reusable Software

When software can be reused, either by multiple users of variations of an application or in updates of an application, important benefits can be realized. These include savings in cost, time, and people. When you use the techniques of domain analysis and domain design, you can write state-of-the-art reusable software.

These are the most important guidelines to follow in writing reusable software:

- Write software that addresses the needs of many people
- Identify old software that is useful
- Design new software to be reusable

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Domain Design:
Writing Reusable Software

- Write software that satisfies the needs of many people.

- Use existing software, if it is useful.

- Design software to be reusable.

9-10

SOFTWARE
PRODUCTIVITY
CONSORTIUM

UNIT 9: SUMMARY OF DOMAIN DESIGN

Summary

We have discussed the exciting field of domain design and software architectures, which represent the cutting edge of software technology.

We covered the following topics:

- Megaprogramming and domain analysis and their relationship to domain design.

- Differences between problems and solutions.

- Domain design products provide solutions to problems defined during domain analysis.

- Architecture, the most important product of domain design, especially software architecture.

- New ways of writing software code to support megaprogamming principles, especially software reuse.

The field of domain design is new and will develop rapidly. Its principles and concepts are of increasing importance in this age of information.

List of Abbreviations and Acronyms

CD	compact disk
CD-ROM	compact disk - read-only memory
CPU	central processing unit
DC	direct current
DOF	domain of focus
DOI	domain of interest
NASA	National Aeronautical and Space Administration
TV	television
URW	United Robot Workers, Inc.
VCR	video cassette recorder

SOFTWARE
PRODUCTIVITY
CONSORTIUM

List of Abbreviations and Acronyms

CD   compact disk
DC   direct current
DOF   domain of focus
DOI   domain of interest
MHz   megahertz
TV   television
VCR   video cassette recorder

Abb-1

SOFTWARE
PRODUCTIVITY
CONSORTIUM

UNIT 10: List of Abbreviations and Acronyms

AARNG	Alaskan Army National Guard
BI	Boom Industries
CD	compact disk
CP	central processor
DOF	domain of focus
DOI	domain of interest
kg	kilogram
PEBOT	Planetary Explorer Robot
URW	United Robot Workers, Inc.